Mellanox
TECHNOLOGIES
Connect. Accelerate. Outperform.™

Mellanox OFED for Linux User Manual

Rev 3.1-1.0.0

www.mellanox.com

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Mellanox Technologies, Ltd.
Hakidma 26
Ofer Industrial Park
Yokneam 2069200
Israel
www.mellanox.com
Tel: +972 (0)74 723 7200
Fax: +972 (0)4 959 3245

Table of Contents

List of Figures

List of Tables

Document Revision History

Release	Date	Description
3.1-1.0.0	September 15, 2015	• Added the following sections: • Section 3.1.13, “Wake-on-LAN (WoL)”, on page 85 • Section 3.1.8.2, “ConnectX-4 ECN”, on page 72 • Section 3.1.9.2, “RSS Verbs Support for ConnectX-4 HCAs”, on page 73 • Section 3.1.14, “Hardware Accelerated 802.1ad VLAN (Q-in-Q Tunneling)”, on page 86 • Updated the following sections: • Section 2.5.1, “Setting up MLNX_OFED YUM Repository”, on page 33 • Section 2.5.2, “Installing MLNX_OFED using the YUM Tool”, on page 35 • Section 2.5.3, “Uninstalling Mellanox OFED using the YUM Tool”, on page 35 • Section 2.6.2, “Installing MLNX_OFED using the apt-get Tool”, on page 36 • Section 2.7.3, “Updating the Device Firmware Automatically upon System Boot”, on page 38 • Section 3.1.4, “Ethtool”, on page 57 • Section 3.1.11.1, “Enable/Disable Flow Steering”, on page 80 • Section 3.1.11.2, “Flow Steering Support”, on page 81 • Section 3.1.11.3, “A0 Static Device Managed Flow Steering”, on page 81 • Section , “Minimal Bandwidth Guarantee (ETS)”, on page 48 • Section 3.2.11.3, “WQ Experimental Family”, on page 177 • Section 3.4.1.2.2, “Configuring SR-IOV for ConnectX-4/Connect-IB”, on page 205
3.0-2.0.0	July 16, 2015	• Added the following sections: • Section 3.4.1.2.2, “Configuring SR-IOV for ConnectX-4/Connect-IB”, on page 205. • Section 3.4.1.2.3, “Note on VFs Initialization”, on page 206

Release	Date	Description
3.0-1.0.1	June 22, 2015	• Added Section 3.1.1.2, “ConnectX-4 Port Type Management”, on page 42
	June 04, 2015	• Added the following sections: • Section 2.3.5, “Driver Load Upon System Boot”, on page 33 • Section 3.1.6, “Ignore Frame Check Sequence (FCS) Errors”, on page 61 • Section 3.1.7.1.1, “RoCE Modes and Device Support”, on page 63 • Section 3.1.7.2.1, “GID Table in sysfs”, on page 63 • Section 3.1.7.2.2, “Setting the RoCE Mode for a QP”, on page 64 • Section 3.1.7.2.3, “Setting RoCE Mode of RDMA_CM Applications”, on page 64 • Section 3.1.7.2.4, “GID Table Example”, on page 64 • Section 3.1.7.5, “RoCE Link Aggregation (RoCE LAG)”, on page 70 • Section 3.2.2.9, “DOS MAD Prevention”, on page 134 • Section 3.2.10, “Resource Domain Experimental Verbs”, on page 175 • Section 3.2.11, “Query Interface Experimental Verbs”, on page 177 • Section 3.3.2, “Sockets Direct Protocol (SDP)”, on page 187 • Section 3.3.2.1, “libsdp.so Library”, on page 187 • Section 3.3.2.2, “Configuring SDP”, on page 187 • Section 3.3.2.3, “Environment Variables”, on page 190 • Section 3.3.2.4, “Converting Socket-based Applications”, on page 190

Release	Date	Description
3.0-1.0.1 (cont.)	June 04, 2015	• Section 3.3.2.5, "BZCopy – Zero Copy Send", on page 196 • Section 3.3.2.6, "Using RDMA for Small Buffers", on page 196 • Section 3.4.1.5.3, "Alias GUID Support in InfiniBand", on page 210 • Section 3.4.1.5.3.1, "Admin VF GUIDs", on page 210 • Section 3.4.1.5.3.2, "On Demand GUIDs", on page 211 • Section 3.4.1.5.3.3, "Alias GUIDs Default Mode", on page 211 • Section 3.4.1.5.3.4, "Initial GUIDs' Values", on page 211 • Section 3.4.1.5.3.5, "Single GUID per VF", on page 211 • Section 3.4.1.9, "Virtualized QoS per VF (Rate Limit per VF)", on page 217 • Section 3.1.12, "WQE Format in MLNX_OFED", on page 84 • Updated the following sections: • Section 1, "Introduction", on page 18 • Section 1.2.2, "mlx5 Driver", on page 20 • Section 2.6.1, "Setting up MLNX_OFED apt-get Repository", on page 36 • Section 3.1.1.3, "Counters", on page 42 • Section 3.1.4, "Ethtool", on page 57 • Section 3.1.7, "RDMA over Converged Ethernet (RoCE)", on page 61 • Removed the following sections: • Configuring The RoCE Mode • Running an - ibv_rc_pingpong Test on the VLAN • Defining Ethernet Priority (PCP in 802.1q Headers) • RoCE Support • Power Management • Adaptive Interrupt Moderation Algorithm • Virtual Guest Tagging (VGT+)

Release	Date	Description
2.4-1.0.0	February 09, 2015	• Updated the ibdump tool in Table 5, “Diagnostic Utilities,” on page 223
	January 15, 2014	• Added the following new sections: • Section 2.7.1, “Updating the Device Online”, on page 37 • Section 3.4.1.8.1, “FDB Status Reporting”, on page 216 • Section 3.1.13, “Adaptive Interrupt Moderation Algorithm”, on page 81 • Section 3.1.9.1.1, “RSS Support for IP Fragments”, on page 73 • Updated Table 1, “ethtool Supported Options,” on page 57 • Updated “`ethtool -K eth<x> [options]`” flag options • Added the following new flags: “`ethtool -s eth<x> speed <SPEED> autoneg off`” and “`ethtool -s eth<x> advertise <N> autoneg on`” • Updated “`port_type_array`” parameter description in Section 3.4.1.2, “Setting Up SR-IOV”, on page 199 • Updated the following sections: • Section 3.1.5, “Checksum Offload”, on page 60 • Section 3.4.3, “VXLAN Hardware Stateless Offloads”, on page 219 • Section 3.4.3.2, “Enabling VXLAN Hardware Stateless Offloads”, on page 219 • Section 5.6, “Performance Related Issues”, on page 234
2.3-2.0.5	January, 2015	• No changes
2.3-2.0.0	November, 2014	• Added the following sections: • Section 3.2.9, “CPU Overhead Distribution”, on page 175 • Section 3.5.1.6, “Extended Error Handling (EEH)”, on page 221 • Section 5.10, “Debugging Related Issues”, on page 237

Release	Date	Description
2.3-1.0.1	September, 2014	• Major restructuring of the User Manual • Updated the following sections: • Section 3.1.11, “Flow Steering”, on page 79 • Added the following sections: • Section 2.6, “Installing MLNX_OFED using apt-get”, on page 36 • Section 2.6.1, “Setting up MLNX_OFED apt-get Repository”, on page 36 • Section 2.6.2, “Installing MLNX_OFED using the apt-get Tool”, on page 36 • Section 2.6.3, “Uninstalling Mellanox OFED using the apt-get Tool”, on page 37 • Section 2.8.2, “Removing Signature from kernel Modules”, on page 39 • Section 3.1.5, “Checksum Offload”, on page 60 • Section 3.1.7.1, “IP Routable RoCE (RoCEv2)”, on page 62 • Section 3.1.7.3, “RoCE Lossless Ethernet Configuration”, on page 65 • Section 3.1.7.3.2, “Configuring SwitchX® Based Switch System”, on page 65 • Section 3.1.6.2.3, “Configuring the RoCE Mode”, on page 58 • Section 3.1.8, “Explicit Congestion Notification (ECN)”, on page 71 • Section 3.2.4, “Secure Host”, on page 138 • Section 3.2.5.1.6, “Dynamic PKey Change”, on page 147 • Section 3.2.7.3, “Memory Region Re-registration”, on page 169 • Section 3.2.7.5, “User-Mode Memory Registration (UMR)”, on page 171 • Section 3.2.7.6, “On-Demand-Paging (ODP)”, on page 171
2.3-1.0.1 (cont.)	September, 2014	• Section 3.4.1.10.1, “Configuring VGT+”, on page 209 • Section 3.5.1, “Reset Flow”, on page 220 • Section 5.1, “General Related Issues”, on page 231 • Section 5.2, “Ethernet Related Issues”, on page 231 • Section 5.3, “InfiniBand Related Issues”, on page 232 • Section 5.4, “InfiniBand/Ethernet Related Issues”, on page 232 • Section 5.5, “Installation Related Issues”, on page 233 • Section 5.6, “Performance Related Issues”, on page 234 • Section 5.7, “SR-IOV Related Issues”, on page 235 • Section 5.8, “PXE (FlexBoot) Related Issues”, on page 235 • Section 5.9, “RDMA Related Issues”, on page 236

About this Manual

This preface provides general information concerning the scope and organization of this User's Manual.

Intended Audience

This manual is intended for system administrators responsible for the installation, configuration, management and maintenance of the software and hardware of VPI (InfiniBand, Ethernet) adapter cards. It is also intended for application developers.

Common Abbreviations and Acronyms

Abbreviation / Acronym	Whole Word / Description
B	(Capital) 'B' is used to indicate size in bytes or multiples of bytes (e.g., 1KB = 1024 bytes, and 1MB = 1048576 bytes)
b	(Small) 'b' is used to indicate size in bits or multiples of bits (e.g., 1Kb = 1024 bits)
FW	Firmware
HCA	Host Channel Adapter
HW	Hardware
IB	InfiniBand
iSER	iSCSI RDMA Protocol
LSB	Least significant *byte*
lsb	Least significant *bit*
MSB	Most significant *byte*
msb	Most significant *bit*
NIC	Network Interface Card
SW	Software
VPI	Virtual Protocol Interconnect
IPoIB	IP over InfiniBand
PFC	Priority Flow Control
PR	Path Record
RDS	Reliable Datagram Sockets
RoCE	RDMA over Converged Ethernet
SDP	Sockets Direct Protocol
SL	Service Level
SRP	SCSI RDMA Protocol
MPI	Message Passing Interface
EoIB	Ethernet over InfiniBand
QoS	Quality of Service

Abbreviation / Acronym	Whole Word / Description
ULP	Upper Level Protocol
VL	Virtual Lane
vHBA	Virtual SCSI Host Bus adapter
uDAPL	User Direct Access Programming Library

Glossary

The following is a list of concepts and terms related to InfiniBand in general and to Subnet Managers in particular. It is included here for ease of reference, but the main reference remains the *InfiniBand Architecture Specification*.

Channel Adapter (CA), Host Channel Adapter (HCA)	An IB device that terminates an IB link and executes transport functions. This may be an HCA (Host CA) or a TCA (Target CA).
HCA Card	A network adapter card based on an InfiniBand channel adapter device.
IB Devices	Integrated circuit implementing InfiniBand compliant communication.
IB Cluster/Fabric/ Subnet	A set of IB devices connected by IB cables.
In-Band	A term assigned to administration activities traversing the IB connectivity only.
Local Identifier (ID)	An address assigned to a port (data sink or source point) by the Subnet Manager, unique within the subnet, used for directing packets within the subnet.
Local Device/Node/ System	The IB Host Channel Adapter (HCA) Card installed on the machine running IBDIAG tools.
Local Port	The IB port of the HCA through which IBDIAG tools connect to the IB fabric.
Master Subnet Manager	The Subnet Manager that is authoritative, that has the reference configuration information for the subnet. See Subnet Manager.
Multicast Forwarding Tables	A table that exists in every switch providing the list of ports to forward received multicast packet. The table is organized by MLID.
Network Interface Card (NIC)	A network adapter card that plugs into the PCI Express slot and provides one or more ports to an Ethernet network.
Standby Subnet Manager	A Subnet Manager that is currently quiescent, and not in the role of a Master Subnet Manager, by agency of the master SM. See Subnet Manager.
Subnet Administrator (SA)	An application (normally part of the Subnet Manager) that implements the interface for querying and manipulating subnet management data.

Subnet Manager (SM)	One of several entities involved in the configuration and control of the an IB fabric.
Unicast Linear Forwarding Tables (LFT)	A table that exists in every switch providing the port through which packets should be sent to each LID.
Virtual Protocol Interconnet (VPI)	A Mellanox Technologies technology that allows Mellanox channel adapter devices (ConnectX®) to simultaneously connect to an InfiniBand subnet and a 10GigE subnet (each subnet connects to one of the adpater ports)

Related Documentation

Document Name	Description
InfiniBand Architecture Specification, Vol. 1, Release 1.2.1	The InfiniBand Architecture Specification that is provided by IBTA
IEEE Std 802.3ae™-2002 (Amendment to IEEE Std 802.3-2002) Document # PDF: SS94996	Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation
Firmware Release Notes for Mellanox adapter devices	See the Release Notes PDF file relevant to your adapter device under `docs/` folder of installed package.
MFT User Manual	Mellanox Firmware Tools User's Manual. See under `docs/` folder of installed package.
MFT Release Notes	Release Notes for the Mellanox Firmware Tools. See under `docs/` folder of installed package.
WinOF User Manual	Mellanox WinOF User Manual describes installation, configuration and operation of Mellanox WinOF driver.

Support and Updates Webpage

Please visit http://www.mellanox.com > Products > InfiniBand/VPI Drivers > Linux SW/Drivers for downloads, FAQ, troubleshooting, future updates to this manual, etc.

1 Introduction

1.1 Overview

Mellanox OFED is a single Virtual Protocol Interconnect (VPI) software stack which operates across all Mellanox network adapter solutions supporting the following uplinks to servers:

- ConnectX®-4:
  - InfiniBand: SDR, QDR, FDR, EDR
  - Ethernet (Beta): 10GigE, 25GigE, 40GigE, 50GigE and 100GigE
- ConnectX®-3/ConnectX®-3 Pro:
  - InfiniBand: SDR, QDR, FDR10, FDR
  - Ethernet: 10GigE, 40GigE and 56GigE[1]
- PCI Express 2.0: 2.5 or 5.0 GT/s
- PCI Express 3.0: 8 GT/s

All Mellanox network adapter cards are compatible with OpenFabrics-based RDMA protocols and software, and are supported with major operating system distributions.

Mellanox OFED is certified with the following products:

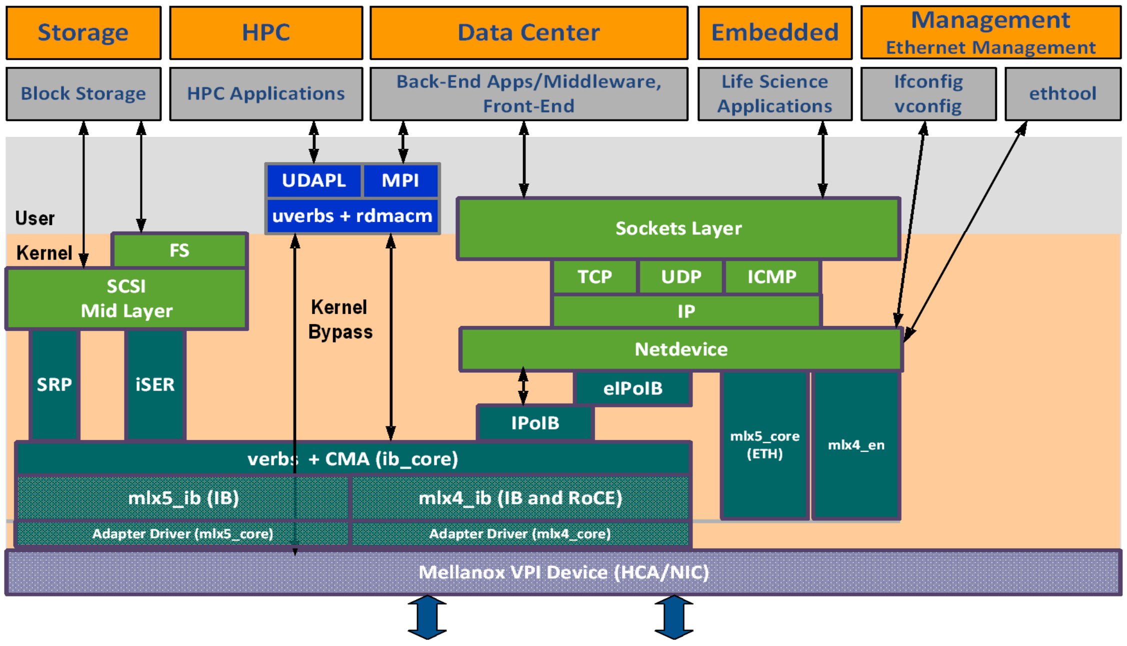
- Mellanox Messaging Accelerator (VMA™) software: Socket acceleration library that performs OS bypass for standard socket based applications.
- Mellanox Unified Fabric Manager (UFM®) software: Powerful platform for managing demanding scale-out computing fabric environments, built on top of the OpenSM industry standard routing engine.
- Fabric Collective Accelerator (FCA) - FCA is a Mellanox MPI-integrated software package that utilizes CORE-Direct technology for implementing the MPI collectives communications.

1. 56 GbE is a Mellanox propriety link speed and can be achieved while connecting a Mellanox adapter cards to Mellanox SX10XX switch series or connecting a Mellanox adapter card to another Mellanox adapter card.

1.2 Stack Architecture

Figure 1 shows a diagram of the Mellanox OFED stack, and how upper layer protocols (ULPs) interface with the hardware and with the kernel and user space. The application level also shows the versatility of markets that Mellanox OFED applies to.

Figure 1: Mellanox OFED Stack for ConnectX® Family Adapter Cards



The following sub-sections briefly describe the various components of the Mellanox OFED stack.

1.2.1 mlx4 VPI Driver

`mlx4` is the low level driver implementation for the ConnectX® family adapters designed by Mellanox Technologies. ConnectX®-2 and ConnectX®-3 adapters can operate as an InfiniBand adapter, or as an Ethernet NIC. The OFED driver supports InfiniBand and Ethernet NIC configurations. To accommodate the supported configurations, the driver is split into the following modules:

mlx4_core

Handles low-level functions like device initialization and firmware commands processing. Also controls resource allocation so that the InfiniBand and Ethernet functions can share the device without interfering with each other.

mlx4_ib

Handles InfiniBand-specific functions and plugs into the InfiniBand midlayer

mlx4_en

A 10/40GigE driver under drivers/net/ethernet/mellanox/mlx4 that handles Ethernet specific functions and plugs into the netdev mid-layer

1.2.2 mlx5 Driver

`mlx5` is the low level driver implementation for the Connect-IB® and ConnectX®-4 adapters designed by Mellanox Technologies. Connect-IB® operates as an InfiniBand adapter whereas and ConnectX®-4 operates as a VPI adapter (Infiniband and Ethernet). The mlx5 driver is comprised of the following kernel modules:

mlx5_core

Acts as a library of common functions (e.g. initializing the device after reset) required by the Connect-IB® and ConnectX®-4 adapter cards. mlx5_core driver also implements the Ethernet interfaces for ConnectX®-4. Unlike mlx4_en/core, mlx5 drivers does not require the mlx5_en module as the Ethernet functionalities are built-in in the mlx5_core module.

mlx5_ib

Handles InfiniBand-specific functions and plugs into the InfiniBand midlayer.

libmlx5

libmlx5 is the provider library that implements hardware specific user-space functionality. If there is no compatibility between the firmware and the driver, the driver will not load and a message will be printed in the dmesg.

The following are the Libmlx5 environment variables:

- MLX5_FREEZE_ON_ERROR_CQE
  - Causes the process to hang in a loop when completion with error which is not flushed with error or retry exceeded occurs/
  - Otherwise disabled
- MLX5_POST_SEND_PREFER_BF
  - Configures every work request that can use blue flame will use blue flame
  - Otherwise - blue flame depends on the size of the message and inline indication in the packet
- MLX5_SHUT_UP_BF
  - Disables blue flame feature
  - Otherwise - do not disable
- MLX5_SINGLE_THREADED
  - All spinlocks are disabled
  - Otherwise - spinlocks enabled
  - Used by applications that are single threaded and would like to save the overhead of taking spinlocks.
- MLX5_CQE_SIZE
  - 64 - completion queue entry size is 64 bytes (default)
  - 128 - completion queue entry size is 128 bytes

- MLX5_SCATTER_TO_CQE
  - Small buffers are scattered to the completion queue entry and manipulated by the driver. Valid for RC transport.
  - Default is 1, otherwise disabled.

1.2.3 Mid-layer Core

Core services include: management interface (MAD), connection manager (CM) interface, and Subnet Administrator (SA) interface. The stack includes components for both user-mode and kernel applications. The core services run in the kernel and expose an interface to user-mode for verbs, CM and management.

1.2.4 ULPs

IPoIB

The IP over IB (IPoIB) driver is a network interface implementation over InfiniBand. IPoIB encapsulates IP datagrams over an InfiniBand connected or datagram transport service. IPoIB pre-appends the IP datagrams with an encapsulation header, and sends the outcome over the InfiniBand transport service. The transport service is Unreliable Datagram (UD) by default, but it may also be configured to be Reliable Connected (RC). The interface supports unicast, multicast and broadcast. For details, see Chapter 3.2.5.1, “IP over InfiniBand (IPoIB)”.

iSER

iSCSI Extensions for RDMA (iSER) extends the iSCSI protocol to RDMA. It permits data to be transferred directly into and out of SCSI buffers without intermediate data copies. For further information, please refer to Chapter 3.3.3, “iSCSI Extensions for RDMA (iSER)”.

SDP

Sockets Direct Protocol (SDP) is a byte-stream transport protocol that provides TCP stream semantics. SDP utilizes InfiniBand's advanced protocol offload capabilities. Because of this, SDP can have lower CPU and memory bandwidth utilization when compared to conventional implementations of TCP, while preserving the TCP APIs and semantics upon which most current network applications depend. For more details, see

SRP

SCSI RDMA Protocol (SRP) is designed to take full advantage of the protocol offload and RDMA features provided by the InfiniBand architecture. SRP allows a large body of SCSI software to be readily used on InfiniBand architecture. The SRP driver—known as the SRP Initiator—differs from traditional low-level SCSI drivers in Linux. The SRP Initiator does not control a local HBA; instead, it controls a connection to an I/O controller—known as the SRP Target—to provide access to remote storage devices across an InfiniBand fabric. The SRP Target resides in an I/O unit and provides storage services. See Chapter 3.3.1, “SCSI RDMA Protocol (SRP)”.

uDAPL

User Direct Access Programming Library (uDAPL) is a standard API that promotes data center application data messaging performance, scalability, and reliability over RDMA interconnects: InfiniBand and RoCE. The uDAPL interface is defined by the DAT collaborative.

This release of the uDAPL reference implementation package for both DAT 1.2 and 2.0 specification is timed to coincide with OFED release of the Open Fabrics (www.openfabrics.org) software stack.

For more information about the DAT collaborative, go to the following site:

http://www.datcollaborative.org

1.2.5 MPI

Message Passing Interface (MPI) is a library specification that enables the development of parallel software libraries to utilize parallel computers, clusters, and heterogeneous networks. Mellanox OFED includes the following MPI implementations over InfiniBand:

- Open MPI – an open source MPI-2 implementation by the Open MPI Project
- OSU MVAPICH – an MPI-1 implementation by Ohio State University

Mellanox OFED also includes MPI benchmark tests such as OSU BW/LAT, Intel MPI Benchmark, and Presta.

1.2.6 InfiniBand Subnet Manager

All InfiniBand-compliant ULPs require a proper operation of a Subnet Manager (SM) running on the InfiniBand fabric, at all times. An SM can run on any node or on an IB switch. OpenSM is an InfiniBand-compliant Subnet Manager, and it is installed as part of Mellanox OFED[1].

1.2.7 Diagnostic Utilities

Mellanox OFED includes the following two diagnostic packages for use by network and datacenter managers:

- ibutils – Mellanox Technologies diagnostic utilities
- infiniband-diags – OpenFabrics Alliance InfiniBand diagnostic tools

1.2.8 Mellanox Firmware Tools

The Mellanox Firmware Tools (MFT) package is a set of firmware management tools for a single InfiniBand node. MFT can be used for:

- Generating a standard or customized Mellanox firmware image
- Burning a firmware image to a single InfiniBand node

MFT includes the following tools:

- mlxburn - provides the following functions:
  - Generation of a standard or customized Mellanox firmware image for burning—in .bin (binary) or .img format
  - Burning an image to the Flash/EEPROM attached to a Mellanox HCA or switch device

1. OpenSM is disabled by default. See Chapter 3.2.2, “OpenSM” for details on enabling it.

  - Querying the firmware version loaded on an HCA board
  - Displaying the VPD (Vital Product Data) of an HCA board
- flint

  This tool burns a firmware binary image or an expansion ROM image to the Flash device of a Mellanox network adapter/bridge/switch device. It includes query functions to the burnt firmware image and to the binary image file.
- Debug utilities

  A set of debug utilities (e.g., itrace, fwtrace, mlxtrace, mlxdump, mstdump, mlxmcg, wqdump, mcra, mlxi2c, i2c, mget_temp, and pckt_drop)

For additional details, please refer to the MFT User's Manual `docs/`.

1.3 Mellanox OFED Package

1.3.1 ISO Image

Mellanox OFED for Linux (MLNX_OFED_LINUX) is provided as ISO images or as a tarball, one per supported Linux distribution and CPU architecture, that includes *source code* and *binary* RPMs, firmware, utilities, and documentation. The ISO image contains an installation script (called `mlnxofedinstall`) that performs the necessary steps to accomplish the following:

- Discover the currently installed kernel
- Uninstall any InfiniBand stacks that are part of the standard operating system distribution or another vendor's commercial stack
- Install the MLNX_OFED_LINUX binary RPMs (if they are available for the current kernel)
- Identify the currently installed InfiniBand HCAs and perform the required firmware updates

1.3.2 Software Components

MLNX_OFED_LINUX contains the following software components:

- Mellanox Host Channel Adapter Drivers
  - mlx5, mlx4 (VPI), which is split into multiple modules:
    - mlx4_core (low-level helper)
    - mlx4_ib (IB)
    - mlx5_ib
    - mlx5_core (includes Ethernet)
    - mlx4_en (Ethernet)
- Mid-layer core
  - Verbs, MADs, SA, CM, CMA, uVerbs, uMADs
- Upper Layer Protocols (ULPs)
  - IPoIB, RDS*, SRP Initiator and SRP

    * **NOTE:** RDS was not tested by Mellanox Technologies.

- MPI
  - Open MPI stack supporting the InfiniBand, RoCE and Ethernet interfaces
  - OSU MVAPICH stack supporting the InfiniBand and RoCE interfaces
  - MPI benchmark tests (OSU BW/LAT, Intel MPI Benchmark, Presta)
- OpenSM: InfiniBand Subnet Manager
- Utilities
  - Diagnostic tools
  - Performance tests
- Firmware tools (MFT)
- Source code for all the OFED software modules (for use under the conditions mentioned in the modules' LICENSE files)
- Documentation

1.3.3 Firmware

The ISO image includes the following firmware items:

- Firmware images (.mlx format) for ConnectX®-3/ConnectX®-3 Pro/Connect-IB® network adapters
- Firmware configuration (.INI) files for Mellanox standard network adapter cards and custom cards

1.3.4 Directory Structure

The ISO image of MLNX_OFED_LINUX contains the following files and directories:

- mlnxofedinstall - This is the MLNX_OFED_LINUX installation script.
- ofed_uninstall.sh - This is the MLNX_OFED_LINUX un-installation script.
- <RPMS folders> - Directory of binary RPMs for a specific CPU architecture.
- firmware/ - Directory of the Mellanox IB HCA firmware images
  (including Boot-over-IB)
- src/ - Directory of the OFED source tarball
- mlnx_add_kernel_support.sh - Script required to rebuild MLNX_OFED_LINUX for customized kernel version on supported Linux Distribution
- RPM based - A script required to rebuild MLNX_OFED_LINUX for customized kernel version on supported RPM based Linux Distribution
- docs/ - Directory of Mellanox OFED related documentation

1.4 Module Parameters

1.4.1 mlx4 Module Parameters

In order to set **`mlx4`** parameters, add the following line(s) to **`/etc/modprobe.conf`**:

```
options mlx4_core parameter=<value>
```

and/or

```
options mlx4_ib   parameter=<value>
```

and/or

```
options mlx4_en   parameter=<value>
```

The following sections list the available `mlx4` parameters.

1.4.1.1 mlx4_ib Parameters

```
sm_guid_assign:             Enable SM alias_GUID assignment if sm_guid_assign > 0
                            (Default: 0) (int)
dev_assign_str[1]:          Map device function numbers to IB device numbers
                            (e.g.'0000:04:00.0-0,002b:1c:0b.a-1,...').
                            Hexadecimal digits for the device function (e.g. 002b:1c:0b.a)
                            and decimal for IB device numbers (e.g. 1). Max supported
                            devices - 32 (string)
```

1. In the current version, this parameter is using decimal number to describe the InfiniBand device and not hexadecimal number as it was in previous versions in order to uniform the mapping of device function numbers to InfiniBand device numbers as defined for other module parameters (e.g. num_vfs and probe_vf).
For example to map mlx4_15 to device function number 04:00.0 in the current version we use `“options mlx4_ib dev_assign_str=04:00.0-15”` as opposed to the previous version in which we used `“options mlx4_ib dev_assign_str=04:00.0-f”`

1.4.1.2 mlx4_core Parameters

```
set_4k_mtu:                 (Obsolete) attempt to set 4K MTU to all ConnectX ports (int)
debug_level:                Enable debug tracing if > 0 (int)
msi_x:                      0 - don't use MSI-X,
                            1 - use MSI-X,
                            >1 - limit number of MSI-X irqs to msi_x (non-SRIOV only) (int)
enable_sys_tune:            Tune the cpu's for better performance (default 0) (int)
block_loopback:             Block multicast loopback packets if > 0 (default: 1) (int)
num_vfs:                    Either a single value (e.g. '5') to define uniform num_vfs
                            value for all devices functions or a string to map device func-
                            tion numbers to their num_vfs values (e.g. '0000:04:00.0-
                            5,002b:1c:0b.a-15').
                            Hexadecimal digits for the device function (e.g. 002b:1c:0b.a)
                            and decimal for num_vfs value (e.g. 15). (string)
probe_vf:                   Either a single value (e.g. '3') to indicate that the Hypervi-
                            sor driver itself should activate this number of VFs for each
                            HCA on the host, or a string to map device function numbers to
                            their probe_vf values (e.g. '0000:04:00.0-3,002b:1c:0b.a-13').
                            Hexadecimal digits for the device function (e.g. 002b:1c:0b.a)
                            and decimal for probe_vf value (e.g. 13). (string)
log_num_mgm_entry_size:     log mgm size, that defines the num of qp per mcg, for example:
                            10 gives 248.range: 7 <= log_num_mgm_entry_size <= 12. To
                            activate device managed flow steering when available, set to -
                            1 (int)
high_rate_steer:            Enable steering mode for higher packet rate (default off)
                            (int)
```

```
fast_drop:                 Enable fast packet drop when no recieve WQEs are posted (int)
enable_64b_cqe_eqe:        Enable 64 byte CQEs/EQEs when the FW supports this if non-zero
                           (default: 1) (int)
log_num_mac:               Log2 max number of MACs per ETH port (1-7) (int)
log_num_vlan:              (Obsolete) Log2 max number of VLANs per ETH port (0-7) (int)
log_mtts_per_seg:          Log2 number of MTT entries per segment (0-7) (default: 0) (int)
port_type_array:           Either pair of values (e.g. '1,2') to define uniform port1/
                           port2 types configuration for all devices functions or a
                           string to map device function numbers to their pair of port
                           types values (e.g. '0000:04:00.0-1;2,002b:1c:0b.a-1;1').
                           Valid port types: 1-ib, 2-eth, 3-auto, 4-N/A
                           If only a single port is available, use the N/A port type for
                           port2 (e.g '1,4').
log_num_qp:                log maximum number of QPs per HCA (default: 19) (int)
log_num_srq:               log maximum number of SRQs per HCA (default: 16) (int)
log_rdmarc_per_qp:         log number of RDMARC buffers per QP (default: 4) (int)
log_num_cq:                log maximum number of CQs per HCA (default: 16) (int)
log_num_mcg:               log maximum number of multicast groups per HCA (default: 13)
                           (int)
log_num_mpt:               log maximum number of memory protection table entries per HCA
                           (default: 19) (int)
log_num_mtt:               log maximum number of memory translation table segments per
                           HCA (default: max(20, 2*MTTs for register all of the host mem-
                           ory limited to 30)) (int)
enable_qos:                Enable Quality of Service support in the HCA (default: off)
                           (bool)
internal_err_reset:        Reset device on internal errors if non-zero (default is 1)
                           (int)
```

1.4.1.3 mlx4_en Parameters

```
inline_thold:              Threshold for using inline data (int)
                           Default and max value is 104 bytes. Saves PCI read operation
                           transaction, packet less then threshold size will be copied to
                           hw buffer directly.
udp_rss:                   Enable RSS for incoming UDP traffic (uint)
                           On by default. Once disabled no RSS for incoming UDP traffic
                           will be done.
pfctx:                     Priority based Flow Control policy on TX[7:0]. Per priority
                           bit mask (uint)
pfcrx:                     Priority based Flow Control policy on RX[7:0]. Per priority
                           bit mask (uint)
```

1.4.2 mlx5 Module Parameters

The mlx5_core module supports a single parameter used to select the profile which defines the number of resources supported. The parameter name for selecting the profile is `prof_sel`.

The supported values for profiles are:

- 0 - for medium resources, medium performance

- 1 - for low resources
- 2 - for high performance (int) (default)

1.5 Device Capabilities

Normally, an application needs to query the device capabilities before attempting to create a resource. It is essential for the application to be able to operate over different devices with different capabilities.

Specifically, when creating a QP, the user needs to specify the maximum number of outstanding work requests that the QP supports. This value should not exceed the queried capabilities. However, even when you specify a number that does not exceed the queried capability, the verbs can still fail since some other factors such as the number of scatter/gather entries requested, or the size of the inline data required, affect the maximum possible work requests. Hence an application should try to decrease this size (halving is a good new value) and retry until it succeeds.

2 Installation

This chapter describes how to install and test the Mellanox OFED for Linux package on a single host machine with Mellanox InfiniBand and/or Ethernet adapter hardware installed.

2.1 Hardware and Software Requirements

Requirements	Description
Platforms	A server platform with an adapter card based on one of the following Mellanox Technologies' InfiniBand HCA devices: • MT4117 ConnectX®-4 Lx (EN) (firmware: fw-ConnectX4Lx) • MT4115 ConnectX®-4 (VPI, IB, EN) (firmware: fw-ConnectX4) • MT4103 ConnectX®-3 Pro (VPI, IB, EN) (firmware: fw-ConnectX3Pro) • MT27508 ConnectX®-3 (VPI, IB, EN) (firmware: fw-ConnectX3) • MT25408 ConnectX®-2 (VPI, IB, EN) (firmware: fw-ConnectX2) • MT4113 Connect-IB® (IB) (firmware: fw-Connect-IB) For the list of supported architecture platforms, please refer to the Mellanox OFED Release Notes file.
Required Disk Space for Installation	1GB
Device ID	For the latest list of device IDs, please visit Mellanox website.
Operating System	Linux operating system. For the list of supported operating system distributions and kernels, please refer to the Mellanox OFED Release Notes file.
Installer Privileges	The installation requires administrator privileges on the target machine.

2.2 Downloading Mellanox OFED

Step 1. Verify that the system has a Mellanox network adapter (HCA/NIC) installed.

The following example shows a system with an installed Mellanox HCA:

```
# lspci -v | grep Mellanox
06:00.0 Network controller: Mellanox Technologies MT27500 Family [ConnectX-3]
Subsystem: Mellanox Technologies Device 0024
```

Step 2. Download the ISO image to your host.

The image's name has the format `MLNX_OFED_LINUX-<ver>-<OS label><CPU arch>.iso`. You can download it from http://www.mellanox.com > Products > Software> InfiniBand Drivers.

Step 3. Use the md5sum utility to confirm the file integrity of your ISO image. Run the following command and compare the result to the value provided on the download page.

```
host1$ md5sum MLNX_OFED_LINUX-<ver>-<OS label>.iso
```

2.3 Installing Mellanox OFED

The installation script, `mlnxofedinstall`, performs the following:

- Discovers the currently installed kernel
- Uninstalls any software stacks that are part of the standard operating system distribution or another vendor's commercial stack
- Installs the MLNX_OFED_LINUX binary RPMs (if they are available for the current kernel)
- Identifies the currently installed InfiniBand and Ethernet network adapters and automatically[1] upgrades the firmware

Usage

```
./mnt/mlnxofedinstall [OPTIONS]
```

The installation script removes all previously installed Mellanox OFED packages and re-installs from scratch. You will be prompted to acknowledge the deletion of the old packages.

Pre-existing configuration files will be saved with the extension ".conf.rpmsave".

- If you need to install Mellanox OFED on an entire (homogeneous) cluster, a common strategy is to mount the ISO image on one of the cluster nodes and then copy it to a shared file system such as NFS. To install on all the cluster nodes, use cluster-aware tools (such as pdsh).
- If your kernel version does not match with any of the offered pre-built RPMs, you can add your kernel version by using the "`mlnx_add_kernel_support.sh`" script located under the docs/ directory.

  On Redhat and SLES distributions with errata kernel installed there is no need to use the mlnx_add_kernel_support.sh script. The regular installation can be performed and weak-updates mechanism will create symbolic links to the MLNX_OFED kernel modules.

  The "`mlnx_add_kernel_support.sh`" script can be executed directly from the mlnxofedinstall script. For further information, please see `'--add-kernel-support'` option below.

  On Ubuntu and Debian distributions drivers installation use Dynamic Kernel Module Support (DKMS) framework. Thus, the drivers' compilation will take place on the host during MLNX_OFED installation.
  Therefore, using "`mlnx_add_kernel_support.sh`" is irrelevant on Ubuntu and Debian distributions.

1. The firmware will not be updated if you run the install script with the '--without-fw-update' option.

Example

The following command will create a MLNX_OFED_LINUX ISO image for RedHat 6.3 under the /tmp directory.

```
# ./MLNX_OFED_LINUX-x.x-x-rhel6.3-x86_64/mlnx_add_kernel_support.sh -m /tmp/MLNX_OFED_-
LINUX-x.x-x-rhel6.3-x86_64/ --make-tgz
Note: This program will create MLNX_OFED_LINUX TGZ for rhel6.3 under /tmp directory.
All Mellanox, OEM, OFED, or Distribution IB packages will be removed.
Do you want to continue?[y/N]:y
See log file /tmp/mlnx_ofed_iso.21642.log

Building OFED RPMs. Please wait...
Removing OFED RPMs...
Created /tmp/MLNX_OFED_LINUX-x.x-x-rhel6.3-x86_64-ext.tgz
```

- The script adds the following lines to `/etc/security/limits.conf` for the userspace components such as MPI:
  - * soft memlock unlimited
  - * hard memlock unlimited
    - These settings set the amount of memory that can be pinned by a user space application to unlimited. If desired, tune the value unlimited to a specific amount of RAM.

For your machine to be part of the InfiniBand/VPI fabric, a Subnet Manager must be running on one of the fabric nodes. At this point, Mellanox OFED for Linux has already installed the OpenSM Subnet Manager on your machine.

2.3.1 Installation Procedure

Step 1. Login to the installation machine as root.

Step 2. Mount the ISO image on your machine.

```
host1# mount -o ro,loop MLNX_OFED_LINUX-<ver>-<OS label>-<CPU arch>.iso /mnt
```

Step 3. Run the installation script.

```
./mnt/mlnxofedinstall
Logs dir: /tmp/MLNX_OFED_LINUX-x.x-x.logs
This program will install the MLNX_OFED_LINUX package on your machine.
Note that all other Mellanox, OEM, OFED, or Distribution IB packages will be removed.
Uninstalling the previous version of MLNX_OFED_LINUX

Starting MLNX_OFED_LINUX-x.x.x installation ...
........
........
Installation finished successfully.

Attempting to perform Firmware update...
Querying Mellanox devices firmware ...
```

In case your machine has the latest firmware, no firmware update will occur and the installation script will print at the end of installation a message similar to the following:

```
Device #1:
----------
  Device Type:      ConnectX3Pro
  Part Number:      MCX354A-FCC_Ax
  Description:      ConnectX-3 Pro VPI adapter card; dual-port
QSFP; FDR IB (56Gb/s) and 40GigE;PCIe3.0 x8 8GT/s;RoHS R6
  PSID:             MT_1090111019
  PCI Device Name:  0000:05:00.0
  Versions:         Current         Available
     FW             2.31.5000       2.31.5000
     PXE            3.4.0224        3.4.0224
  Status:        Up to date
```

In case your machine has an unsupported network adapter device, no firmware update will occur and the error message below will be printed. Please contact your hardware vendor for help on firmware updates.

Error message:

```
Device #1:
----------
  Device:        0000:05:00.0
  Part Number:
  Description:
  PSID:          MT_0DB0110010
  Versions:      Current        Available
     FW          2.9.1000       N/A
  Status:        No matching image found
```

Step 4. Reboot the machine **if** the installation script performed firmware updates to your network adapter hardware. Otherwise, restart the driver by running: "`/etc/init.d/openibd restart`"

Step 5. (InfiniBand only) Run the `hca_self_test.ofed` utility to verify whether or not the InfiniBand link is up. The utility also checks for and displays additional information such as:

- HCA firmware version
- Kernel architecture
- Driver version
- Number of active HCA ports along with their states
- Node GUID

For more details on `hca_self_test.ofed`, see the file `hca_self_test.readme` under `docs/`.

After the installer completes, information about the Mellanox OFED installation such as prefix, kernel version, and installation parameters can be retrieved by running the command `/etc/infiniband/info`.

Most of the Mellanox OFED components can be configured or reconfigured after the installation by modifying the relevant configuration files. See the relevant chapters in this manual for details.

The list of the modules that will be loaded automatically upon boot can be found in the `/etc/infiniband/openib.conf` file.

2.3.2 Installation Results

Software	• Most of MLNX_OFED packages are installed under the “/usr” directory except for the following packages which are installed under the “/opt” directory: • openshmem, bupc, fca and ibutils • The kernel modules are installed under • /lib/modules/\`uname -r\`/updates on SLES and Fedora Distributions • /lib/modules/\`uname -r\`/extra/mlnx-ofa_kernel on RHEL and other RedHat like Distributions • /lib/modules/\`uname -r\`/updates/dkms/ on Ubuntu

Firmware	• The firmware of existing network adapter devices will be updated if the following two conditions are fulfilled:
	  • The installation script is run in default mode; that is, without the option ‘--without-fw-update’
	  • The firmware version of the adapter device is older than the firmware version included with the Mellanox OFED ISO image
	  Note: If an adapter’s Flash was originally programmed with an Expansion ROM image, the automatic firmware update will also burn an Expansion ROM image.
	• In case your machine has an unsupported network adapter device, no firmware update will occur and the error message below will be printed.
	`-I- Querying device ...`
	`-E- Can't auto detect fw configuration file: ...`
	Please contact your hardware vendor for help on firmware updates.

2.3.3 Installation Logging

While installing MLNX_OFED, the install log for each selected package will be saved in a separate log file.

The path to the directory containing the log files will be displayed after running the installation script in the following format: `"Logs dir: /tmp/MLNX_OFED_LINUX-<version>.<PID>.logs"`.

Example:

```
Logs dir: /tmp/MLNX_OFED_LINUX-x.x-x.logs
```

2.3.4 openibd Script

As of MLNX_OFED v2.2-1.0.0 the `openibd` script supports pre/post start/stop scripts:

This can be controlled by setting the variables below in the `/etc/infiniband/openibd.conf` file.

```
OPENIBD_PRE_START
OPENIBD_POST_START
OPENIBD_PRE_STOP
OPENIBD_POST_STOP
```

Example:

```
OPENIBD_POST_START=/sbin/openibd_post_start.sh
```

2.3.5 Driver Load Upon System Boot

Upon system boot, the Mellanox drivers will be loaded automatically.

➢ ***To prevent automatic load of the Mellanox drivers upon system boot:***

Step 1. Add the following lines to the "`/etc/modprobe.d/mlnx.conf`" file.

```
blacklist mlx4_core
blacklist mlx4_en
blacklist mlx5_core
blacklist mlx5_ib
```

Step 2. Set "`ONBOOT=no`" in the "`/etc/infiniband/openib.conf`" file.

2.3.6 mlnxofedinstall Return Codes

The table below lists the `mlnxofedinstall` script return codes and their meanings.

Return Code	Meaning
0	The Installation ended successfully
1	The installation failed
2	No firmware was found for the adapter device
22	Invalid parameter
28	Not enough free space
171	Not applicable to this system configuration. This can occur when the required hardware is not present on the system.
172	Prerequisites are not met. For example, missing the required software installed or the hardware is not configured correctly.
173	Failed to start the `mst` driver

2.4 Uninstalling Mellanox OFED

Use the script `/usr/sbin/ofed_uninstall.sh` to uninstall the Mellanox OFED package. The script is part of the `ofed-scripts` RPM.

2.5 Installing MLNX_OFED using YUM

This type of installation is applicable to RedHat/OEL, Fedora, XenServer Operating Systems.

2.5.1 Setting up MLNX_OFED YUM Repository

Step 1. Log into the installation machine as root.

Step 2. Mount the ISO image on your machine and copy its content to a shared location in your network.

```
# mount -o ro,loop MLNX_OFED_LINUX-<ver>-<OS label>-<CPU arch>.iso /mnt
```

You can download it from http://www.mellanox.com > Products > Software> InfiniBand Drivers.

Step 3. Download and install Mellanox Technologies GPG-KEY:

The key can be downloaded via the following link:

http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox

```
# wget http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox
--2014-04-20 13:52:30--  http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox
Resolving www.mellanox.com... 72.3.194.0
Connecting to www.mellanox.com|72.3.194.0|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1354 (1.3K) [text/plain]
Saving to: ?RPM-GPG-KEY-Mellanox?

100%[=================================================>] 1,354       --.-K/s   in 0s

2014-04-20 13:52:30 (247 MB/s) - ?RPM-GPG-KEY-Mellanox? saved [1354/1354]
```

Step 4. Install the key.

```
# sudo rpm --import RPM-GPG-KEY-Mellanox
warning: rpmts_HdrFromFdno: Header V3 DSA/SHA1 Signature, key ID 6224c050: NOKEY
Retrieving key from file:///repos/MLNX_OFED/<MLNX_OFED file>/RPM-GPG-KEY-Mellanox
Importing GPG key 0x6224C050:
 Userid: "Mellanox Technologies (Mellanox Technologies - Signing Key v2) <support@mel-
lanox.com>"
 From  : /repos/MLNX_OFED/<MLNX_OFED file>/RPM-GPG-KEY-Mellanox
Is this ok [y/N]:
```

Step 5. Check that the key was successfully imported.

```
# rpm -q gpg-pubkey --qf '%{NAME}-%{VERSION}-%{RELEASE}\t%{SUMMARY}\n' | grep Mellanox
gpg-pubkey-a9e4b643-520791ba    gpg(Mellanox Technologies <support@mellanox.com>)
```

Step 6. Create a yum repository configuration file called `"/etc/yum.repos.d/mlnx_ofed.repo"` with the following content:.

```
[mlnx_ofed]
name=MLNX_OFED Repository
baseurl=file:///<path to extracted MLNX_OFED package>
enabled=1
gpgkey=file:///<path to the downloaded key RPM-GPG-KEY-Mellanox>
gpgcheck=1
```

Step 7. Check that the repository was successfully added.

```
# yum repolist
Loaded plugins: product-id, security, subscription-manager
This system is not registered to Red Hat Subscription Management. You can use subscrip-
tion-manager to register.
repo id          repo name                                       status
mlnx_ofed        MLNX_OFED Repository                            108
rpmforge         RHEL 6Server - RPMforge.net - dag               4,597

repolist: 8,351
```

2.5.2 Installing MLNX_OFED using the YUM Tool

After setting up the YUM repository for MLNX_OFED package, perform the following:

Step 1. View the available package groups by invoking:

```
# yum search mlnx-ofed-
mlnx-ofed-all.noarch : MLNX_OFED all installer package
mlnx-ofed-basic.noarch : MLNX_OFED basic installer package
mlnx-ofed-guest.noarch : MLNX_OFED guest installer package
mlnx-ofed-hypervisor.noarch : MLNX_OFED hypervisor installer package
mlnx-ofed-vma.noarch : MLNX_OFED vma installer package
mlnx-ofed-vma-eth.noarch : MLNX_OFED vma-eth installer package
mlnx-ofed-vma-vpi.noarch : MLNX_OFED vma-vpi installer package
```

Where:

mlnx-ofed-all	Installs all available packages in MLNX_OFED.
mlnx-ofed-basic	Installs basic packages required for running Mellanox cards.
mlnx-ofed-guest	Installs packages required by guest OS.
mlnx-ofed-hypervisor	Installs packages required by hypervisor OS.
mlnx-ofed-vma	Installs packages required by VMA.
mlnx-ofed-vma-eth	Installs packages required by VMA to work over Ethernet.
mlnx-ofed-vma-vpi	Installs packages required by VMA to support VPI.

Step 2. Install the desired group.

```
 # yum install mlnx-ofed-all
 Loaded plugins: langpacks, product-id, subscription-manager
 Resolving Dependencies
 --> Running transaction check
 ---> Package mlnx-ofed-all.noarch 0:3.1-0.1.2 will be installed
 --> Processing Dependency: kmod-isert = 1.0-OFED.3.1.0.1.2.1.g832a737.rhel7u1 for pack-
age: mlnx-ofed-all-3.1-0.1.2.noarch
 .................
 .................
   qperf.x86_64 0:0.4.9-9
   rds-devel.x86_64 0:2.0.7-1.12
   rds-tools.x86_64 0:2.0.7-1.12
   sdpnetstat.x86_64 0:1.60-26
   srptools.x86_64 0:1.0.2-12

 Complete!
```

Installing MLNX_OFED using the YUM tool does not automatically update the firmware.

To update the firmware to the version included in MLNX_OFED package, you can either:

- Run `# yum install mlnx-fw-updater`

  or
- Update the firmware to the latest version available on Mellanox Technologies' Web site as described in section Section 2.7, "Updating Firmware After Installation", on page 37.

2.5.3 Uninstalling Mellanox OFED using the YUM Tool

Use the script `/usr/sbin/ofed_uninstall.sh` to uninstall the Mellanox OFED package. The script is part of the ofed-scripts RPM.

2.6 Installing MLNX_OFED using apt-get

This type of installation is applicable to Debian and Ubuntu Operating Systems.

2.6.1 Setting up MLNX_OFED apt-get Repository

Step 1. Log into the installation machine as root.

Step 2. Extract the MLNX_OFED pacakge on a shared location in your network.

You can download it from http://www.mellanox.com > Products > Software> InfiniBand Drivers.

Step 3. Create an apt-get repository configuration file called "`/etc/apt/sources.list.d/mlnx-_ofed.list`" with the following content:

```
deb file:/<path to extracted MLNX_OFED package> ./
```

Step 4. Download and install Mellanox Technologies GPG-KEY.

```
# wget -qO - http://www.mellanox.com/downloads/ofed/RPM-GPG-KEY-Mellanox | sudo apt-key 
add -
```

Step 5. Check that the key was successfully imported.

```
# apt-key list
pub   1024D/A9E4B643 2013-08-11
uid   Mellanox Technologies <support@mellanox.com>
sub   1024g/09FCC269 2013-08-11
```

Step 6. Update the apt-get cache.

```
# sudo apt-get update
```

2.6.2 Installing MLNX_OFED using the apt-get Tool

After setting up the apt-get repository for MLNX_OFED package, perform the following:

Step 1. View the available package groups by invoking:

```
# apt-cache search mlnx-ofed-
    mlnx-ofed-all
    mlnx-ofed-basic
    mlnx-ofed-vma
    mlnx-ofed-vma-eth
    mlnx-ofed-vma-vpi
```

Where:

```
mlnx-ofed-all            MLNX_OFED all installer package.
mlnx-ofed-basic          MLNX_OFED basic installer package.
mlnx-ofed-vma            MLNX_OFED vma installer package.
mlnx-ofed-vma-eth        MLNX_OFED vma-eth installer package.
mlnx-ofed-vma-vpi        MLNX_OFED vma-vpi installer package.
```

Step 2. Install the desired group.

```
apt-get install '<group name>'
```

Example:

```
apt-get install mlnx-ofed-all
```

Installing MLNX_OFED using the "apt-get" tool does not automatically update the firmware.

To update the firmware to the version included in MLNX_OFED package, run:

```
# apt-get install mlnx-fw-updater
```

2.6.3 Uninstalling Mellanox OFED using the apt-get Tool

Use the script `/usr/sbin/ofed_uninstall.sh` to uninstall the Mellanox OFED package. The script is part of the `ofed-scripts` RPM.

2.7 Updating Firmware After Installation

The firmware can be updated in one of the following methods.

2.7.1 Updating the Device Online

To update the device online on the machine from Mellanox site, use the following command line:

```
mlxfwmanager --online -u -d <device>
```

Example:

```
mlxfwmanager --online -u -d 0000:09:00.0
Querying Mellanox devices firmware ...

Device #1:
----------

Device Type:      ConnectX3
Part Number:      MCX354A-FCA_A2-A4
Description:      ConnectX-3 VPI adapter card; dual-port QSFP; FDR IB (56Gb/s) and
40GigE;           PCIe3.0 x8 8GT/s; RoHS R6
PSID:             MT_1020120019
PCI Device Name:  0000:09:00.0
Port1 GUID:       0002c9000100d051
Port2 MAC:        0002c9000002
Versions:         Current        Available
FW                2.32.5000      2.33.5000

Status:           Update required
---------
Found 1 device(s) requiring firmware update. Please use -u flag to perform the update.
```

2.7.2 Updating the Device Manually

In case you ran the `mlnxofedinstall` script with the '`--without-fw-update`' option or you are using an OEM card and now you wish to (manually) update firmware on your adapter card(s), you need to perform the steps below. The following steps are also appropriate in case you wish to burn newer firmware that you have downloaded from Mellanox Technologies' Web site (http://www.mellanox.com > Support > Firmware Download).

Step 1. Get the device’s PSID.

```
mlxfwmanager_pci | grep PSID
PSID:            MT_1210110019
```

Step 2. Download the firmware BIN file from the Mellanox website or the OEM website.

Step 3. Burn the firmware.

```
mlxfwmanager_pci -i <fw_file.bin>
```

Step 4. Reboot your machine after the firmware burning is completed.

2.7.3 Updating the Device Firmware Automatically upon System Boot

As of MLNX_OFED v3.1-x.x.x, firmware can be automatically updated upon system boot.

The firmware update package (`mlnx-fw-updater`) is installed in the “`/opt/mellanox/mlnx-fw-updater`” folder, and openibd service script can invoke the firmware update process if requested on boot.

If the firmware is updated, the following message is printed to the system’s standard logging file:

```
fw_updater: Firmware was updated. Please reboot your system for the changes to take 
effect.
```

Otherwise, the following message is printed:

```
fw_updater: Didn't detect new devices with old firmware.
```

Please note, this feature is disabled by default. To enable the automatic firmware update upon system boot, set the following parameter to “`yes`” “`RUN_FW_UPDATER_ONBOOT=yes`” in the openibd service configuration file “`/etc/infiniband/openib.conf`”.

You can opt to exclude a list of devices from the automatic firmware update procedure. To do so, edit the configurations file “`/opt/mellanox/mlnx-fw-updater/mlnx-fw-updater.conf`” and provide a comma separated list of PCI devices to exclude from the firmware update.

Example:

```
MLNX_EXCLUDE_DEVICES="00:05.0,00:07.0"
```

2.8 UEFI Secure Boot

All kernel modules included in MLNX_OFED for RHEL7 and SLES12 are signed with x.509 key to support loading the modules when Secure Boot is enabled.

2.8.1 Enrolling Mellanox's x.509 Public Key On your Systems

In order to support loading MLNX_OFED drivers when an OS supporting Secure Boot boots on a UEFI-based system with Secure Boot enabled, the Mellanox x.509 public key should be added to the UEFI Secure Boot key database and loaded onto the system key ring by the kernel.

Follow these steps below to add the Mellanox's x.509 public key to your system:

Prior to adding the Mellanox's x.509 public key to your system, please make sure:
- the 'mokutil' package is installed on your system
- the system is booted in UEFI mode

Step 1. Download the x.509 public key.

```
# wget http://www.mellanox.com/downloads/ofed/mlnx_signing_key_pub.der
```

Step 2. Add the public key to the MOK list using the mokutil utility.

You will be asked to enter and confirm a password for this MOK enrollment request.

```
# mokutil --import mlnx_signing_key_pub.der
```

Step 3. Reboot the system.

The pending MOK key enrollment request will be noticed by `shim.efi` and it will launch `Mok-Manager.efi` to allow you to complete the enrollment from the UEFI console. You will need to enter the password you previously associated with this request and confirm the enrollment. Once done, the public key is added to the MOK list, which is persistent. Once a key is in the MOK list, it will be automatically propagated to the system key ring and subsequent will be booted when the UEFI Secure Boot is enabled.

To see what keys have been added to the system key ring on the current boot, install the 'keyutils' package and run: `#keyctl list %:.system_keyring`

2.8.2 Removing Signature from kernel Modules

The signature can be removed from a signed kernel module using the 'strip' utility which is provided by the 'binutils' package.

```
# strip -g my_module.ko
```

The strip utility will change the given file without saving a backup. The operation can be undo only by resigning the kernel module. Hence, we recommend backing up a copy prior to removing the signature.

➢ ***To remove the signature from the MLNX_OFED kernel modules:***

Step 1. Remove the signature.

```
# rpm -qa | grep -E "kernel-ib|mlnx-ofa_kernel|iser|srp|knem" | xargs rpm -ql | grep
"\.ko$" | xargs strip -g
```

After the signature has been removed, a massage as the below will no longer be presented upon module loading:

```
"Request for unknown module key 'Mellanox Technologies signing key:
61feb074fc7292f958419386ffdd9d5ca999e403' err -11"
```

However, please note that a similar message as the following will still be presented:

```
"my_module: module verification failed: signature and/or required key missing - taint-
ing kernel"
```

This message is presented once, only for each boot for the first module that either has no signature or whose key is not in the kernel key ring. So it's much easier to miss this message. You won't see it on repeated tests where you unload and reload a kernel module until you reboot. There is no way to eliminate this message.

Step 2. Update the initramfs on RHEL systems with the stripped modules.

```
mkinitrd /boot/initramfs-$(uname -r).img $(uname -r) --force
```

2.9 Performance Tuning

For further information on Linux performance, please refer to the Performance Tuning Guide for Mellanox Network Adapters.

3 Features Overview and Configuration

3.1 Ethernet Network

3.1.1 Interface

3.1.1.1 ConnectX-3/ConnectX-3 Pro Port Type Management

ConnectX®-3/ConnectX®-3 Pro ports can be individually configured to work as InfiniBand or Ethernet ports. By default both ConnectX ports are initialized as InfiniBand ports. If you wish to change the port type use the `connectx_port_config` script after the driver is loaded.

Running "`/sbin/connectx_port_config -s`" will show current port configuration for all ConnectX devices.

Port configuration is saved in the file: `/etc/infiniband/connectx.conf`. This saved configuration is restored at driver restart only if restarting via "`/etc/init.d/openibd restart`".

Possible port types are:

- eth – Ethernet
- ib – InfiniBand
- auto – Link sensing mode - Detect port type based on the attached network type. If no link is detected, the driver retries link sensing every few seconds.

The port link type can be configured for each device in the system at run time using the "`/sbin/connectx_port_config`" script. This utility will prompt for the PCI device to be modified (if there is only one it will be selected automatically).

In the next stage the user will be prompted for the desired mode for each port. The desired port configuration will then be set for the selected device.

This utility also has a non-interactive mode:

```
/sbin/connectx_port_config [[-d|--device <PCI device ID>] -c|--conf <port1,port2>]"
```

Auto Sensing enables the NIC to automatically sense the link type (InfiniBand or Ethernet) based on the link partner and load the appropriate driver stack (InfiniBand or Ethernet).
For example, if the first port is connected to an InfiniBand switch and the second to Ethernet switch, the NIC will automatically load the first switch as InfiniBand and the second as Ethernet.

Upon driver start up:

1. Sense the adapter card's port type:

   If a valid cable or module is connected (QSFP, SFP+, or SFP with EEPROM in the cable/module):

   - Set the port type to the sensed link type (IB/Ethernet)

     Otherwise:

   - Set the port type as default (Ethernet)

During driver run time:

- Sense a link every 3 seconds if no link is sensed/detected
- If sensed, set the port type as sensed

3.1.1.2 ConnectX-4 Port Type Management

ConnectX®-4 ports can be individually configured to work as InfiniBand or Ethernet ports. By default both ConnectX®-4 VPI ports are initialized as InfiniBand ports. If you wish to change the port type, use the `mlxconfig` script after the driver is loaded.

For further information on how to set the port type in ConnectX®-4, please refer to the MFT User Manual (www.mellanox.con --> Products --> Software --> InfiniBand/VPI Software --> Linux SW/Drivers)

3.1.1.3 Counters

Counters are used to provide information about how well an operating system, an application, a service, or a driver is performing. The counter data helps determine system bottlenecks and fine-tune the system and application performance. The operating system, network, and devices provide counter data that an application can consume to provide users with a graphical view of how well the system is performing.

The counter index is a QP attribute given in the QP context. Multiple QPs may be associated with the same counter set, If multiple QPs share the same counter its value represents the cumulative total.

- ConnectX®-3 supports 127 different counters which are allocated as follow:
  - 4 counters reserved for PF - 2 counters for each port
  - 2 counters reserved for VF - 1 counter for each port
  - All other counters if exist are allocated by demand
- RoCE counters are available only through sysfs located under:
  - # /sys/class/infiniband/mlx4_*/ports/*/counters/
  - # /sys/class/infiniband/mlx4_*/ports/*/counters_ext/
- Physical Function can also read Virtual Functions' port counters through sysfs located under:
  - # /sys/class/net/eth*/vf*/statistics/

To display the network device Ethernet statistics, you can run:

```
Ethtool -S <devname>
```

Counter	Description
rx_packets	Total packets successfully received.
rx_bytes	Total bytes in successfully received packets.
rx_multicast_packets	Total multicast packets successfully received.
rx_broadcast_packets	Total broadcast packets successfully received.
rx_errors	Number of receive packets that contained errors preventing them from being deliverable to a higher-layer protocol.
rx_dropped	Number of receive packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol.
rx_length_errors	Number of received frames that were dropped due to an error in frame length

Counter	Description
rx_over_errors	Number of received frames that were dropped due to hard-ware port receive buffer overflow
rx_crc_errors	Number of received frames with a bad CRC that are not runts, jabbers, or alignment errors
rx_jabbers	Number of received frames with a length greater than MTU octets and a bad CRC
rx_in_range_length_error	Number of received frames with a length/type field value in the (decimal) range [1500:46] (42 is also counted for VLAN-tagged frames)
rx_out_range_length_error	Number of received frames with a length/type field value in the (decimal) range [1535:1501]
tx_packets	Total packets successfully transmitted.
tx_bytes	Total bytes in successfully transmitted packets.
tx_multicast_packets	Total multicast packets successfully transmitted.
tx_broadcast_packets	Total broadcast packets successfully transmitted.
tx_errors	Number of frames that failed to transmit
tx_dropped	Number of transmitted frames that were dropped
rx_prio_<i>_packets	Total packets successfully received with priority i.
rx_prio_<i>_bytes	Total bytes in successfully received packets with priority i.
rx_novlan_packets	Total packets successfully received with no VLAN priority.
rx_novlan_bytes	Total bytes in successfully received packets with no VLAN priority.
tx_prio_<i>_packets	Total packets successfully transmitted with priority i.
tx_prio_<i>_bytes	Total bytes in successfully transmitted packets with priority i.
tx_novlan_packets	Total packets successfully transmitted with no VLAN prior-ity.
tx_novlan_bytes	Total bytes in successfully transmitted packets with no VLAN priority.
rx_pause[1]	The total number of PAUSE frames received from the far-end port.
rx_pause_duration[1]	The total time in microseconds that far-end port was requested to pause transmission of packets.
rx_pause_transition[1]	The number of receiver transitions from XON state (paused) to XOFF state (non-paused)
tx_pause[1]	The total number of PAUSE frames sent to the far-end port
tx_pause_duration[1]	The total time in microseconds that transmission of packets has been paused
tx_pause_transition[1]	The number of transmitter transitions from XON state (paused) to XOFF state (non-paused)
vport_rx_unicast_packets	Unicast packets received successfully

Counter	Description
vport_rx_unicast_bytes	Unicast packet bytes received successfully
vport_rx_multicast_packets	Multicast packets received successfully
vport_rx_multicast_bytes	Multicast packet bytes received successfully
vport_rx_broadcast_packets	Broadcast packets received successfully
vport_rx_broadcast_bytes	Broadcast packet bytes received successfully
vport_rx_dropped	Received packets discarded due to luck of software receive buffers (WQEs). Important indication to weather RX completion routines are keeping up with hardware ingress packet rate
vport_rx_filtered	Received packets dropped due to packet check that failed. For example: Incorrect VLAN, incorrect Ethertype, unavailable queue/QP or loopback prevention
vport_tx_unicast_packets	Unicast packets sent successfully
vport_tx_unicast_bytes	Unicast packet bytes sent successfully
vport_tx_multicast_packets	Multicast packets sent successfully
vport_tx_multicast_bytes	Multicast packet bytes sent successfully
vport_tx_broadcast_packets	Broadcast packets sent successfully
vport_tx_broadcast_bytes	Broadcast packet bytes sent successfully
vport_tx_dropped	Packets dropped due to transmit errors
rx_lro_aggregated	Number of packets processed by the LRO mechanism
rx_lro_flushed	Number of offloaded packets the LRO mechanism passed to kernel
rx_lro_no_desc	LRO mechanism has no room to receive packets from the adapter. In normal condition, it should not increase
rx_alloc_failed	Number of times failed preparing receive descriptor
rx_csum_good	Number of packets received with good checksum
rx_csum_none	Number of packets received with no checksum indication
tx_chksum_offload	Number of packets transmitted with checksum offload
tx_queue_stopped	Number of times transmit queue suspended
tx_wake_queue	Number of times transmit queue resumed
tx_timeout	Number of times transmitter timeout
xmit_more	Number of times doorbell was not triggered due to skb xmit more.
tx_tso_packets	Number of packet that were aggregated
rx<i>_packets	Total packets successfully received on ring i
rx<i>_bytes	Total bytes in successfully received packets on ring i.
tx<i>_packets	Total packets successfully transmitted on ring i.
tx<i>_bytes	Total bytes in successfully transmitted packets on ring i.

1. Pause statistics can be divided into “prio_<i>”, depending on PFC configuration set.

3.1.1.4 Persistent Naming

To avoid network interface renaming after boot or driver restart use the `"/etc/udev/rules.d/70-persistent-net.rules"` file.

- Example for Ethernet interfaces:

```
 # PCI device 0x15b3:0x1003 (mlx4_core)
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:02:c9:fa:c3:50",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:02:c9:fa:c3:51",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth2"
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:02:c9:e9:56:a1",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth3"
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:02:c9:e9:56:a2",
ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="eth*", NAME="eth4"
```

- Example for IPoIB interfaces:

```
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{dev_id}=="0x0", ATTR{type}=="32",
NAME="ib0"
 SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{dev_id}=="0x1", ATTR{type}=="32",
NAME="ib1"
```

3.1.2 Quality of Service (QoS)

Quality of Service (QoS) (supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only) is a mechanism of assigning a priority to a network flow (socket, rdma_cm connection) and manage its guarantees, limitations and its priority over other flows. This is accomplished by mapping the user's priority to a hardware TC (traffic class) through a 2/3 stages process. The TC is assigned with the QoS attributes and the different flows behave accordingly

3.1.2.1 Mapping Traffic to Traffic Classes

Mapping traffic to TCs (supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only) consists of several actions which are user controllable, some controlled by the application itself and others by the system/network administrators.

The following is the general mapping traffic to Traffic Classes flow:

1. The application sets the required Type of Service (ToS).
2. The ToS is translated into a Socket Priority (`sk_prio`).
3. The `sk_prio` is mapped to a User Priority (UP) by the system administrator (some applications set `sk_prio` directly).
4. The UP is mapped to TC by the network/system administrator.
5. TCs hold the actual QoS parameters

QoS can be applied on the following types of traffic. However, the general QoS flow may vary among them:

- **Plain Ethernet** - Applications use regular inet sockets and the traffic passes via the kernel Ethernet driver
- **RoCE** - Applications use the RDMA API to transmit using QPs
- **Raw Ethernet QP** - Application use VERBs API to transmit using a Raw Ethernet QP

3.1.2.2 Plain Ethernet Quality of Service Mapping

Applications use regular inet sockets and the traffic passes via the kernel Ethernet driver.

The following is the Plain Ethernet QoS mapping flow:

1. The application sets the ToS of the socket using `setsockopt` (`IP_TOS`, value).
2. ToS is translated into the `sk_prio` using a fixed translation:

```
TOS 0 <=> sk_prio 0
TOS 8 <=> sk_prio 2
TOS 24 <=> sk_prio 4
TOS 16 <=> sk_prio 6
```

3. The Socket Priority is mapped to the UP:
   - If the underlying device is a VLAN device, `egress_map` is used controlled by the `vconfig` command. This is per VLAN mapping.
   - If the underlying device is not a VLAN device, the `tc` command is used. In this case, even though `tc` manual states that the mapping is from the `sk_prio` to the TC number, the `mlx-4_en` driver interprets this as a `sk_prio` to UP mapping.
     Mapping the sk_prio to the UP is done by using `tc_wrap.py -i <dev name> -u 0,1,2,3,4,5,6,7`
4. The UP is mapped to the TC as configured by the `mlnx_qos` tool or by the `lldpad` daemon if DCBX is used.

Socket applications can use `setsockopt` (`SK_PRIO`, value) to directly set the `sk_prio` of the socket. In this case the ToS to `sk_prio` fixed mapping is not needed. This allows the application and the administrator to utilize more than the 4 values possible via ToS.

In case of VLAN interface, the UP obtained according to the above mapping is also used in the VLAN tag of the traffic

3.1.2.3 RoCE Quality of Service Mapping

Applications use RDMA-CM API to create and use QPs.

The following is the RoCE QoS mapping flow:

1. The application sets the ToS of the QP using the `rdma_set_option` option (`RDMA_OP-TION_ID_TOS`, value).
2. ToS is translated into the Socket Priority (`sk_prio`) using a fixed translation:

```
TOS 0 <=> sk_prio 0
TOS 8 <=> sk_prio 2
TOS 24 <=> sk_prio 4
TOS 16 <=> sk_prio 6
```

3. The Socket Priority is mapped to the User Priority (UP) using the `tc` command.
   - In case of a VLAN device where the parent real device is used for the purpose of this mapping
   - If the underlying device is a VLAN device, and the parent real device was not used for the mapping, the VLAN device's egress_map is used

- On Kernels 3.13 and higher or Kernels that ported the functionality of enabled access to the VLAN device egress map (through vlan_dev_get_egress_qos_mask() call in if_vlan.h)

4. The UP is mapped to the TC as configured by the `mlnx_qos` tool or by the `lldpad` daemon if DCBX is used.

With RoCE, there can only be 4 predefined ToS values for the purpose of QoS mapping.

3.1.2.4 Raw Ethernet QP Quality of Service Mapping

Applications open a Raw Ethernet QP using VERBs directly.

The following is the RoCE QoS mapping flow:

1. The application sets the UP of the Raw Ethernet QP during the INIT to RTR state transition of the QP:
   - Sets `qp_attrs.ah_attrs.sl = up`
   - Calls `modify_qp` with `IB_QP_AV` set in the mask
2. The UP is mapped to the TC as configured by the `mlnx_qos` tool or by the `lldpad` daemon if DCBX is used

When using Raw Ethernet QP mapping, the TOS/sk_prio to UP mapping is lost.

Performing the Raw Ethernet QP mapping forces the QP to transmit using the given UP. If packets with VLAN tag are transmitted, UP in the VLAN tag will be overwritten with the given UP.

3.1.2.5 Map Priorities with tc_wrap.py/mlnx_qos

Network flow that can be managed by QoS attributes is described by a User Priority (UP). A user's `sk_prio` is mapped to UP which in turn is mapped into TC.

- Indicating the UP
  - When the user uses `sk_prio`, it is mapped into a UP by the `'tc'` tool. This is done by the `tc_wrap.py` tool which gets a list of <= 16 comma separated UP and maps the `sk_prio` to the specified UP.
    For example, `tc_wrap.py -ieth0 -u 1,5` maps `sk_prio 0` of `eth0` device to UP 1 and `sk_prio 1` to UP 5.
  - Setting `set_egress_map` in VLAN, maps the `skb_priority` of the VLAN to a `vlan_qos`. The `vlan_qos` is represents a UP for the VLAN device.
  - In RoCE, `rdma_set_option` with `RDMA_OPTION_ID_TOS` could be used to set the UP
  - When creating QPs, the `sl` field in `ibv_modify_qp` command represents the UP
- Indicating the TC
  - After mapping the `skb_priority` to UP, one should map the UP into a TC. This assigns the user priority to a specific hardware traffic class. In order to do that, `mlnx_qos` should

be used. `mlnx_qos` gets a list of a mapping between UPs to TCs. For example, `mlnx_qos -ieth0 -p 0,0,0,0,1,1,1,1` maps UPs 0-3 to `TC0`, and Ups 4-7 to `TC1`.

3.1.2.6 Quality of Service Properties

The different QoS properties (supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only) that can be assigned to a TC are:

- Strict Priority (see "Strict Priority")
- Minimal Bandwidth Guarantee (ETS) (see "Minimal Bandwidth Guarantee (ETS)")
- Rate Limit (see "Rate Limit")

Strict Priority

When setting a TC's transmission algorithm to be 'strict', then this TC has absolute (strict) priority over other TC strict priorities coming before it (as determined by the TC number: TC 7 is highest priority, TC 0 is lowest). It also has an absolute priority over non strict TCs (ETS).

This property needs to be used with care, as it may easily cause starvation of other TCs.

A higher strict priority TC is always given the first chance to transmit. Only if the highest strict priority TC has nothing more to transmit, will the next highest TC be considered.

Non strict priority TCs will be considered last to transmit.

This property is extremely useful for low latency low bandwidth traffic. Traffic that needs to get immediate service when it exists, but is not of high volume to starve other transmitters in the system.

Minimal Bandwidth Guarantee (ETS)

After servicing the strict priority TCs, the amount of bandwidth (BW) left on the wire may be split among other TCs according to a minimal guarantee policy.

If, for instance, TC0 is set to 80% guarantee and TC1 to 20% (the TCs sum must be 100), then the BW left after servicing all strict priority TCs will be split according to this ratio.

Since this is a minimal guarantee, there is no maximum enforcement. This means, in the same example, that if TC1 did not use its share of 20%, the reminder will be used by TC0.

ETS is configured using the mlnx_qos tool ("mlnx_qos") which allows you to:

- Assign a transmission algorithm to each TC (strict or ETS)
- Set minimal BW guarantee to ETS TCs

Usage:

```
mlnx_qos -i [options]
```

Rate Limit

Rate limit defines a maximum bandwidth allowed for a TC. Please note that 10% deviation from the requested values is considered acceptable.

3.1.2.7 Quality of Service Tools

mlnx_qos

`mlnx_qos` is a centralized tool used to configure QoS features of the local host. It communicates directly with the driver thus does not require setting up a DCBX daemon on the system.

The `mlnx_qos` tool enables the administrator of the system to:

- Inspect the current QoS mappings and configuration

  The tool will also display maps configured by TC and `vconfig set_egress_map` tools, in order to give a centralized view of all QoS mappings.
- Set UP to TC mapping
- Assign a transmission algorithm to each TC (strict or ETS)
- Set minimal BW guarantee to ETS TCs
- Set rate limit to TCs

For unlimited ratelimit set the ratelimit to 0.

Usage:

```
mlnx_qos -i <interface> [options]
```

Options:

```
    --version              show program's version number and exit
    -h, --help             show this help message and exit
    -p LIST, --prio_tc=LIST
                           maps UPs to TCs. LIST is 8 comma seperated TC numbers.
                           Example: 0,0,0,0,1,1,1,1 maps UPs 0-3 to TC0, and UPs
                           4-7 to TC1
    -s LIST, --tsa=LIST    Transmission algorithm for each TC. LIST is comma
                           seperated algorithm names for each TC. Possible
                           algorithms: strict, etc. Example: ets,strict,ets sets
                           TC0,TC2 to ETS and TC1 to strict. The rest are
                           unchanged.
    -t LIST, --tcbw=LIST   Set minimal guaranteed %BW for ETS TCs. LIST is comma
                           seperated percents for each TC. Values set to TCs that
                           are not configured to ETS algorithm are ignored, but
                           must be present. Example: if TC0,TC2 are set to ETS,
                           then 10,0,90 will set TC0 to 10% and TC2 to 90%.
                           Percents must sum to 100.
    -r LIST, --ratelimit=LIST
                           Rate limit for TCs (in Gbps). LIST is a comma
                           seperated Gbps limit for each TC. Example: 1,8,8 will
                           limit TC0 to 1Gbps, and TC1,TC2 to 8 Gbps each.
    -i INTF, --interface=INTF
                           Interface name
    -a                     Show all interface's TCs
```

Get Current Configuration:

```
# mlnx_qos -i <interface>
tc: 0 ratelimit: unlimited, tsa: strict
         up:  0
                 skprio: 0
                 skprio: 1
                 skprio: 2 (tos: 8)
                 skprio: 3
                 skprio: 4 (tos: 24)
                 skprio: 5
                 skprio: 6 (tos: 16)
                 skprio: 7
                 skprio: 8
                 skprio: 9
                 skprio: 10
                 skprio: 11
                 skprio: 12
                 skprio: 13
                 skprio: 14
                 skprio: 15
         up:  1
         up:  2
         up:  3
         up:  4
         up:  5
         up:  6
         up:  7
```

Set ratelimit. 3Gbps for tc0 4Gbps for tc1 and 2Gbps for tc2:

```
# mlnx_qos -i <interface> -p 0,1,2 -r 3,4,2
tc: 0 ratelimit: 3 Gbps, tsa: strict
         up:  0
                 skprio: 0
                 skprio: 1
                 skprio: 2 (tos: 8)
                 skprio: 3
                 skprio: 4 (tos: 24)
                 skprio: 5
                 skprio: 6 (tos: 16)
                 skprio: 7
                 skprio: 8
                 skprio: 9
                 skprio: 10
                 skprio: 11
                 skprio: 12
                 skprio: 13
                 skprio: 14
                 skprio: 15
         up:  3
         up:  4
         up:  5
         up:  6
         up:  7
tc: 1 ratelimit: 4 Gbps, tsa: strict
         up:  1
tc: 2 ratelimit: 2 Gbps, tsa: strict
         up:  2
```

Configure QoS. map UP 0,7 to tc0, 1,2,3 to tc1 and 4,5,6 to tc 2. set tc0,tc1 as ets and tc2 as strict. divide ets 30% for tc0 and 70% for tc1:

```
# mlnx_qos -i <interface> -s ets,ets,strict -p 0,1,1,1,2,2,2 -t 30,70tc: 0 ratelimit: 3 Gbps, 
tsa: ets, bw: 30%
         up:  0
                 skprio: 0
                 skprio: 1
                 skprio: 2 (tos: 8)
                 skprio: 3
                 skprio: 4 (tos: 24)
                 skprio: 5
                 skprio: 6 (tos: 16)
                 skprio: 7
                 skprio: 8
                 skprio: 9
                 skprio: 10
                 skprio: 11
                 skprio: 12
                 skprio: 13
                 skprio: 14
                 skprio: 15
```

```
  up:  7
tc: 1 ratelimit: 4 Gbps, tsa: ets, bw: 70%
         up:  1
         up:  2
         up:  3
tc: 2 ratelimit: 2 Gbps, tsa: strict
         up:  4
         up:  5
         up:  6
```

tc and tc_wrap.py

The 'tc' tool is used to setup `sk_prio` to UP mapping, using the `mqprio` queue discipline.

In kernels that do not support `mqprio` (such as 2.6.34), an alternate mapping is created in `sysfs`. The 'tc_wrap.py' tool will use either the `sysfs` or the 'tc' tool to configure the `sk_prio` to UP mapping.

Usage:

```
tc_wrap.py -i <interface> [options]
```

Options:

```
--version            show program's version number and exit
-h, --help           show this help message and exit
-u SKPRIO_UP, --skprio_up=SKPRIO_UP  maps sk_prio to UP. LIST is <=16 comma separated
                                     UP. index of element is sk_prio.
-i INTF, --interface=INTF            Interface name
```

Set skprio 0-2 to UP0, and skprio 3-7 to UP1 on Ethernet Interface

```
h-qa-le014:~ # tc_wrap.py -i eth4 -u 0,0,0,1,1,1,1,1
UP 0
        skprio: 0
        skprio: 1
        skprio: 2 (tos: 8)
        skprio: 8
        skprio: 9
        skprio: 10
        skprio: 11
        skprio: 12
        skprio: 13
        skprio: 14
        skprio: 15
```

```
UP 1
        skprio: 3
        skprio: 4 (tos: 24)
        skprio: 5
        skprio: 6 (tos: 16)
        skprio: 7
UP 2
UP 3
UP 4
UP 5
UP 6
UP 7
```

Additional Tools

tc tool compiled with the `sch_mqprio` module is required to support kernel v2.6.32 or higher. This is a part of `iproute2` package v2.6.32-19 or higher. Otherwise, an alternative custom sysfs interface is available.

- `mlnx_qos tool` (package: ofed-scripts) requires python >= 2.5
- `tc_wrap.py` (package: ofed-scripts) requires python >= 2.5

3.1.3 Quantized Congestion Notification (QCN)

Supported in ConnectX-3 and ConnectX-3 Pro only.

Congestion control is used to reduce packet drops in lossy environments and mitigate congestion spreading and resulting victim flows in lossless environments.

The Quantized Congestion Notification (QCN) IEEE standard (802.1Qau) provides congestion control for long-lived flows in limited bandwidth-delay product Ethernet networks. It is part of the IEEE Data Center Bridging (DCB) protocol suite, which also includes ETS, PFC, and DCBX. QCN in conducted at L2, and is targeted for hardware implementations. QCN applies to all Ethernet packets and all transports, and both the host and switch behavior is detailed in the standard.

QCN user interface allows the user to configure QCN activity. QCN configuration and retrieval of information is done by the `mlnx_qcn` tool. The command interface provides the user with a set of changeable attributes, and with information regarding QCN's counters and statistics. All parameters and statistics are defined per port and priority. QCN command interface is available if and only the hardware supports it.

3.1.3.1 QCN Tool - mlnx_qcn

mlnx_qcn is a tool used to configure QCN attributes of the local host. It communicates directly with the driver thus does not require setting up a DCBX daemon on the system.

The mlnx_qcn enables the user to:

- Inspect the current QCN configurations for a certain port sorted by priority
- Inspect the current QCN statistics and counters for a certain port sorted by priority
- Set values of chosen QCN parameters

Usage:

```
mlnx_qcn -i <interface> [options]
```

Options:

--version	Show program's version number and exit
-h, --help	Show this help message and exit
-i INTF, --interface=INTF	Interface name
-g TYPE, --get_type=TYPE	Type of information to get statistics/parameters
--rpg_enable=RPG_ENABLE_LIST	Set value of rpg_enable according to priority, use spaces between values and -1 for unknown values.
--rppp_max_rps=RPPP_MAX_RPS_LIST	Set value of rppp_max_rps according to priority, use spaces between values and -1 for unknown values.
--rpg_time_reset=RPG_TIME_RESET_LIST	Set value of rpg_time_reset according to priority, use spaces between values and -1 for unknown values.
--rpg_byte_reset=RPG_BYTE_RESET_LIST	Set value of rpg_byte_reset according to priority, use spaces between values and -1 for unknown values.
--rpg_threshold=RPG_THRESHOLD_LIST	Set value of rpg_threshold according to priority, use spaces between values and -1 for unknown values.
--rpg_max_rate=RPG_MAX_RATE_LIST	Set value of rpg_max_rate according to priority, use spaces between values and -1 for unknown values.
--rpg_ai_rate=RPG_AI_RATE_LIST	Set value of rpg_ai_rate according to priority, use spaces between values and -1 for unknown values.
--rpg_hai_rate=RPG_HAI_RATE_LIST	Set value of rpg_hai_rate according to priority, use spaces between values and -1 for unknown values.
--rpg_gd=RPG_GD_LIST	Set value of rpg_gd according to priority, use spaces between values and -1 for unknown values.
--rpg_min_dec_fac=RPG_MIN_DEC_FAC_LIST	Set value of rpg_min_dec_fac according to priority, use spaces between values and -1 for unknown values.
--rpg_min_rate=RPG_MIN_RATE_LIST	Set value of rpg_min_rate according to priority, use spaces between values and -1 for unknown values.
--cndd_state_machine=CNDD_STATE_MACHINE_LIST	Set value of cndd_state_machine according to priority, use spaces between values and -1 for unknown values.

➢ ***To get QCN current configuration sorted by priority:***

```
mlnx_qcn -i eth2 -g parameters
```

➢ ***To show QCN's statistics sorted by priority:***

```
mlnx_qcn -i eth2 -g statistics
```

Example output when running mlnx_qcn -i eth2 -g parameters:

```
priority 0:
        rpg_enable: 0
        rppp_max_rps: 1000
        rpg_time_reset: 1464
        rpg_byte_reset: 150000
        rpg_threshold: 5
        rpg_max_rate: 40000
        rpg_ai_rate: 10
        rpg_hai_rate: 50
        rpg_gd: 8
        rpg_min_dec_fac: 2
        rpg_min_rate: 10
        cndd_state_machine: 0
priority 1:
        rpg_enable: 0
        rppp_max_rps: 1000
        rpg_time_reset: 1464
        rpg_byte_reset: 150000
        rpg_threshold: 5
        rpg_max_rate: 40000
        rpg_ai_rate: 10
        rpg_hai_rate: 50
        rpg_gd: 8
        rpg_min_dec_fac: 2
        rpg_min_rate: 10
        cndd_state_machine: 0
.............................
.............................
priority 7:
        rpg_enable: 0
        rppp_max_rps: 1000
        rpg_time_reset: 1464
        rpg_byte_reset: 150000
        rpg_threshold: 5
        rpg_max_rate: 40000
        rpg_ai_rate: 10
        rpg_hai_rate: 50
        rpg_gd: 8
        rpg_min_dec_fac: 2
        rpg_min_rate: 10
        cndd_state_machine: 0
```

Setting QCN Configuration

Setting the QCN parameters, requires updating its value for each priority. '-1' indicates no change in the current value.

Example for setting 'rp g_enable' in order to enable QCN for priorities 3, 5, 6:

```
mlnx_qcn -i eth2 --rpg_enable=-1 -1 -1 1 -1 1 1 -1
```

Example for setting 'rpg_hai_rate' for priorities 1, 6, 7:

```
mlnx_qcn -i eth2 --rpg_hai_rate=60 -1 -1 -1 -1 -1 60 60
```

3.1.4 Ethtool

ethtool is a standard Linux utility for controlling network drivers and hardware, particularly for wired Ethernet devices. It can be used to:

- Get identification and diagnostic information
- Get extended device statistics
- Control speed, duplex, auto-negotiation and flow control for Ethernet devices
- Control checksum offload and other hardware offload features
- Control DMA ring sizes and interrupt moderation

The following are the ethtool supported options:

Table 1 - ethtool Supported Options

Options	Description

ethtool -i eth<x>	Checks driver and device information.
	For example:
	`#> ethtool -i eth2`
	`driver: mlx4_en (MT_0DD0120009_CX3)`
	`version: 2.1.6 (Aug 2013)`
	`firmware-version: 2.30.3000`
	`bus-info: 0000:1a:00.0`

ethtool -k eth<x>	Queries the stateless offload status.
ethtool -c eth<x>	Queries interrupt coalescing settings.
ethtool -C eth<x> adaptive-rx on|off	**Note:** Supported in ConnectX®-3/ConnectX®-3 Pro cards only. Enables/disables adaptive interrupt moderation. By default, the driver uses adaptive interrupt moderation for the receive path, which adjusts the moderation time to the traffic pattern. For further information, please refer to Adaptive Interrupt Moderation section.
ethtool -C eth<x> [pkt-rate-low N] [pkt-rate-high N] [rx-usecs-low N] [rx-usecs-high N]	**Note:** Supported in ConnectX®-3/ConnectX®-3 Pro cards only. Sets the values for packet rate limits and for moderation time high and low values. For further information, please refer to Adaptive Interrupt Moderation section.

Table 1 - ethtool Supported Options

Options	Description
ethtool -C eth<x> [rx-usecs N] [rx-frames N]	Sets the interrupt coalescing setting. rx-frames will be enforced immediately, rx-usecs will be enforced only when adaptive moderation is disabled. **Note:** usec settings correspond to the time to wait after the *last* packet is sent/received before triggering an interrupt.
ethtool -K eth<x> [rx on|off] [tx on|off] [sg on|off] [tso on|off] [lro on|off] [gro on|off] [gso on|off] [rxvlan on|off] [txvlan on|off] [ntu-ple on/off] [rxhash on/off] [rx-all on/off] [rx-fcs on/off]	Sets the stateless offload status. TCP Segmentation Offload (TSO), Generic Segmentation Offload (GSO): increase outbound throughput by reducing CPU overhead. It works by queuing up large buffers and let-ting the network interface card split them into separate packets. Large Receive Offload (LRO): increases inbound through-put of high-bandwidth network connections by reducing CPU overhead. It works by aggregating multiple incoming packets from a single stream into a larger buffer before they are passed higher up the networking stack, thus reducing the number of packets that have to be processed. LRO is avail-able in kernel versions < 3.1 for untagged traffic. Hardware VLAN insertion Offload (txvlan): When enabled, the sent VLAN tag will be inserted into the packet by the hardware. **Note:** LRO will be done whenever possible. Otherwise GRO will be done. Generic Receive Offload (GRO) is available throughout all kernels. Hardware VLAN Striping Offload (rxvlan): When enabled received VLAN traffic will be stripped from the VLAN tag by the hardware. RX FCS (rx-fcs): Keeps FCS field in the received packets. RX FCS validation (rx-all): Ignores FCS validation on the received packets. **Note:** The flags below are supported in ConnectX®-3/Con-nectX®-3 Pro cards only: `[rxvlan on|off] [txvlan on|off] [ntuple on/off] [rxhash on/off] [rx-all on/off] [rx-fcs on/off]`
ethtool -a eth<x>	**Note:** Supported in ConnectX®-3/ConnectX®-3 Pro cards only. Queries the pause frame settings.
ethtool -A eth<x> [rx on|off] [tx on|off]	**Note:** Supported in ConnectX®-3/ConnectX®-3 Pro cards only. Sets the pause frame settings.

Table 1 - ethtool Supported Options

Options	Description
ethtool -g eth<x>	Queries the ring size values.
ethtool -G eth<x> [rx <N>] [tx <N>]	Modifies the rings size.
ethtool -p|--identify eth<x> <LED duration>	Allows users to identify interface's physical port by turning the ports LED on for a number of seconds. **Note:** The limit for the LED duration is 65535 seconds.
ethtool -S eth<x>	Obtains additional device statistics.
ethtool -t eth<x>	Performs a self diagnostics test.
ethtool -s eth<x> msglvl [N]	Changes the current driver message level.
ethtool -T eth<x>	**Note:** Supported in ConnectX®-3/ConnectX®-3 Pro cards only. Shows time stamping capabilities
ethtool -l eth<x>	Shows the number of channels
ethtool -L eth<x> [rx <N>] [tx <N>]	Sets the number of channels **Note:** For ConnectX®-4 cards, use `ethtool -L eth<x> combined <N>` to set both RX and TX channels.
etthtool -m|--dump-module-eeprom eth<x> [raw on|off] [hex on|off] [offset N] [length N]	Queries/Decodes the cable module eeprom information.
ethtool --show-priv-flags eth<x>	Shows driver private flags and their states (ON/OFF) The private flag is: • qcn_disable_32_14_4_e The flags below indicate the flow steering current configuration and limits. • mlx4_flow_steering_ethernet_l2 • mlx4_flow_steering_ipv4 • mlx4_flow_steering_tcp For further information, refer to Flow Steering section. The flags below are related to *Ignore Frame Check Sequence*, and they are active when `ethtool -k` does not support them: • orx-fcs • orx-all
ethtool --set-priv-flags eth<x> <priv flag> <on/off>	Enables/disables driver feature matching the given private flag.
ethtool -s eth<x> speed <SPEED> autoneg off	Changes the link speed to requested <SPEED>. To check the supported speeds, run `ethtool eth<x>`. **NOTE**: `<autoneg off>` does not set autoneg OFF, it only hints the driver to set a specific speed.

Table 1 - ethtool Supported Options

Options	Description
ethtool -s eth<x> advertise <N> autoneg on	Changes the advertised link modes to requested link modes <N> To check the link modes' hex values, run `<man ethtool>` and to check the supported link modes, run `ethtoo eth<x>` **NOTE**: <autoneg on> only sends a hint to the driver that the user wants to modify advertised link modes and not speed.
ethtool -X eth<x> equal a b c...	Sets the receive flow hash indirection table.
ethtool -x eth<x>	Retrieves the receive flow hash indirection table.

3.1.5 Checksum Offload

MLNX_OFED supports the following Receive IP/L4 Checksum Offload modes:

- CHECKSUM_UNNECESSARY: By setting this mode the driver indicates to the Linux Networking Stack that the hardware successfully validated the IP and L4 checksum so the Linux Networking Stack does not need to deal with IP/L4 Checksum validation.

  Checksum Unnecessary is passed to the OS when all of the following are true:
  - Ethtool -k <DEV> shows rx-checksumming: on
  - Received TCP/UDP packet and both IP checksum and L4 protocol checksum are correct.
- CHECKSUM_COMPLETE: When the checksum validation cannot be done or fails, the driver still reports to the OS the calculated by hardware checksum value. This allows accelerating checksum validation in Linux Networking Stack, since it does not have to calculate the whole checksum including payload by itself.

  Checksum Complete is passed to OS when all of the following are true:
  - Ethtool -k <DEV> shows rx-checksumming: on
  - Using ConnectX®-3, firmware version 2.31.7000 and up
  - Received IpV4/IpV6 non TCP/UDP packet

The ingress parser of the ConnectX®-3-Pro card comes by default without checksum offload support for non TCP/UDP packets.

To change that, please set the value of the module parameter `ingress_parser_mode` in mlx4_core to 1.

In this mode IPv4/IPv6 non TCP/UDP packets will be passed up to the protocol stack with CHECKSUM_COMPLETE tag.

In this mode of the ingress parser, the following features are unavailable:
- NVGRE stateless offloads
- VXLAN stateless offloads
- RoCE v2 (RoCE over UDP)

Change the default behavior only if non tcp/udp is very common.

- CHECKSUM_NONE: By setting this mode the driver indicates to the Linux Networking Stack that the hardware failed to validate the IP or L4 checksum so the Linux Networking Stack must calculate and validate the IP/L4 Checksum.

  Checksum None is passed to OS for all other cases.

3.1.6 Ignore Frame Check Sequence (FCS) Errors

Supported in ConnectX-3 Pro and ConnectX-4 only.

Upon receiving packets, the packets go through a checksum validation process for the FCS field. If the validation fails, the received packets are dropped.

When FCS is enabled (disabled by default), the device does not validate the FCS field even if the field is invalid.

It is not recommended to enable FCS.

For further information on how to enable/disable FCS, please refer to Table 1, "ethtool Supported Options," on page 57

3.1.7 RDMA over Converged Ethernet (RoCE)

Remote Direct Memory Access (RDMA) is the remote memory management capability that allows server-to-server data movement directly between application memory without any CPU involvement. RDMA over Converged Ethernet (RoCE) is a mechanism to provide this efficient data transfer with very low latencies on lossless Ethernet networks. With advances in data center convergence over reliable Ethernet, ConnectX® EN with RoCE uses the proven and efficient RDMA transport to provide the platform for deploying RDMA technology in mainstream data center application at 10GigE and 40GigE link-speed. ConnectX® EN with its hardware offload support takes advantage of this efficient RDMA transport (InfiniBand) services over Ethernet to deliver ultra-low latency for performance-critical and transaction intensive applications such as financial, database, storage, and content delivery networks.

RoCE encapsulates IB transport in one of the following Ethernet packet

- RoCEv1 - dedicated ether type (0x8915)
- RoCEv2 - UDP and dedicated UDP port (4791)

When working with RDMA applications over Ethernet link layer the following points should be noted:

- The presence of a Subnet Manager (SM) is not required in the fabric. Thus, operations that require communication with the SM are managed in a different way in RoCE. This does not affect the API but only the actions such as joining multicast group, that need to be taken when using the API
- Since LID is a layer 2 attribute of the InfiniBand protocol stack, it is not set for a port and is displayed as zero when querying the port
- With RoCE, the alternate path is not set for RC QP and therefore APM is not supported
- Since the SM is not present, querying a path is impossible. Therefore, the path record structure must be filled with the relevant values before establishing a connection. Hence, it is recommended working with RDMA-CM to establish a connection as it takes care of filling the path record structure
- VLAN tagged Ethernet frames carry a 3-bit priority field. The value of this field is derived from the IB SL field by taking the 3 least significant bits of the SL field
- RoCE traffic is not shown in the associated Ethernet device's counters since it is off-loaded by the hardware and does not go through Ethernet network driver. RoCE traffic

is counted in the same place where InfiniBand traffic is counted; /sys/class/infiniband/<device>/ports/<port number>/counters/

3.1.7.1 IP Routable RoCE (RoCEv2)

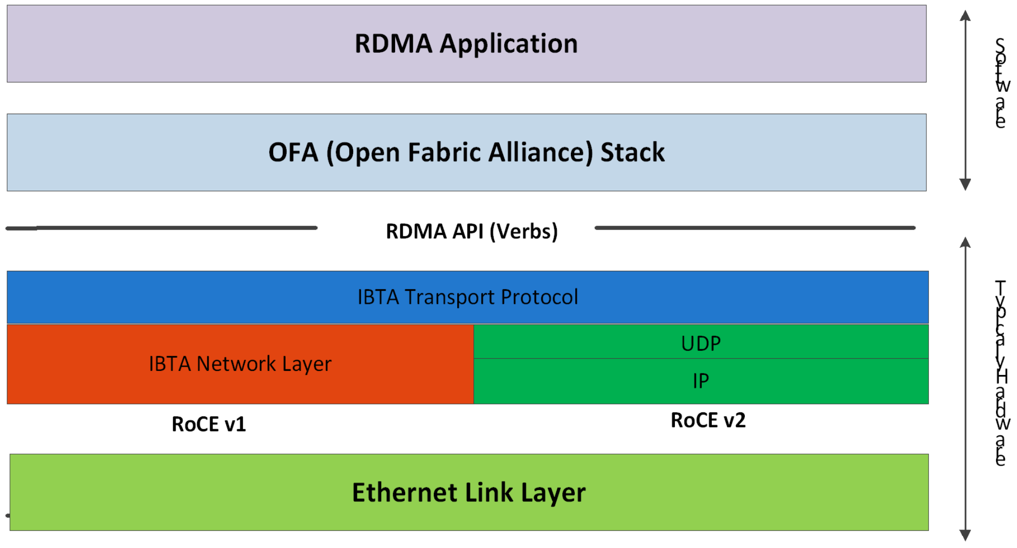
RoCE v2 is supported only in ConnectX®-3 Pro and ConnectX®-4 adapter cards.
In ConnectX®-4, RoCE v2 is at Beta level.

A straightforward extension of the RoCE protocol enables traffic to operate in IP layer 3 environments. This capability is obtained via a simple modification of the RoCE packet format. Instead of the GRH used in RoCE, IP routable RoCE packets carry an IP header which allows traversal of IP L3 Routers and a UDP header (RoCEv2 only) that serves as a stateless encapsulation layer for the RDMA Transport Protocol Packets over IP.

The proposed RoCEv2 packets use a well-known UDP destination port value that unequivocally distinguishes the datagram. Similar to other protocols that use UDP encapsulation, the UDP source port field is used to carry an opaque flow-identifier that allows network devices to implement packet forwarding optimizations (e.g. ECMP) while staying agnostic to the specifics of the protocol header format.

Furthermore, since this change exclusively affects the packet format on the wire, and due to the fact that with RDMA semantics packets are generated and consumed below the AP, applications can seamlessly operate over any form of RDMA service, in a completely transparent way.

Figure 2: RoCEv1 and RoCEv2 Protocol Stack



3.1.7.1.1 RoCE Modes and Device Support

RoCE packets can be of the following types:

- RoCEv1- over Ethernet. No IP header
- RoCEv2- over IP/UDP

While ConnectX®-3 supports only RoCEv1, ConnectX®-3 Pro supports both RoCEv1 and RoCEv2. The RoCE mode can be set using the `'roce_mode'` parameter.

- If set to '0', the driver associates all GID indexes to RoCEv1
- If set to '2', the driver associates all GID indexes to RoCEv2
  (supported in ConnectX-3 Pro as of firmware v2.32.5100)
- If set to '4', the driver associates all GID indexes to RoCEv1 and RoCEv2, a single entry for each RoCE version. (supported in ConnectX-3 Pro as of firmware v2.34.5000)

ConnectX®-4 supports both RoCEv1 and RoCEv2, (currently at beta level). By default, the driver associates all GID indexes to RoCEv1 and RoCEv2, thus, a single entry for each RoCE version.

3.1.7.2 GID Table Population

GID table entries are created whenever an IP address is configured on one of the Ethernet devices of the NIC's ports. Each entry in the GID table for RoCE ports has the following fields:

- GID value
- GID type
- Network device

For ports on devices that support two RoCE modes (ConnectX®-3 Pro) the table will be occupied with two GID, both with the same GID value but with different types. The Network device in an entry is the Ethernet device with the IP address that GID is associated with. The GID format can be of 2 types, IPv4 and IPv6. IPv4 GID is a IPv4-mapped IPv6 address while IPv6 GID is the IPv6 address itself. Layer 3 header for packets associated with IPv4 GIDs will be IPv4 (for RoCEv2) and IPv6/GRH for packets associated with IPv6 GIDs and IPv4 GIDs for RoCEv1.

The number of entries in the GID table is equal to N[1,2](K+1) where N is the number of IP addresses that are assigned to all network devices associated with the port including VLAN devices, alias devices and bonding masters (for active slaves only). Link local IPv6 addresses are excluded from this count since the GID for them is always preset (the default GIDs) at the beginning of each table. K is the number of the supported RoCE types. Since the number of entries in the hardware is limited to 128 for each port, it is important to understand the limitations on N.

3.1.7.2.1 GID Table in sysfs

GID table is exposed to user space via sysfs

- GID values can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gids/{index}
```

1. When the mode of the device is RoCEv1/RoCEv2, each entry in the GID table occupies 2 entries in the hardware. In other modes, each entry in the GID table occupies a single entry in the hardware.
2. In multifunction configuration, the PF gets 16 entries in the hardware while each VF gets 112/F where F is the number of virtual functions on the port. If 112/F is not an integer, some functions will have 1 less entries than others. **Note** that when F is larger than 56, some VFs will get only one entry in the GID table.

- GID type can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gid_attrs/types/{index}
```

- GID net_device can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gid_attrs/ndevs/{index}
```

3.1.7.2.2 Setting the RoCE Mode for a QP

Setting RoCE mode for devices that support two RoCE modes is different for RC/UC QPs (connected QP types) and UD QP.

To modify an RC/UC QP (connected QP) from INIT to RTR, an Address Vector (AV) must be given. The AV, among other attributes, should specify the index of the port's GID table for the source GID of the QP. The GID type in that index will be used to set the RoCE type of the QP.

To modify UD QPs, the value of the module parameter `'ud_gid_type'` must be used to set the RoCE mode for all UD QPs on the device.

3.1.7.2.3 Setting RoCE Mode of RDMA_CM Applications

RDMA_CM interface requires only the active side of the peer to pass the IP address of the passive side. The RDMA_CM decides upon the source GID to be use and obtains it from the GID table. Since more than one instance of the GID value is possible, the lookup should be also according to the GID type. The type to use for the lookup is defined as a global value of the RDMA_CM module. Changing the value of the GID type for the GID table lookups is done using configfs as shown in the example below:

```
mount -t configfs none /sys/kernel/config
cd /sys/kernel/config/rdma_cm
mkdir mlx4_0
cd mlx4_0
echo RoCE V2 > default_roce_mode (*)
cd ..
rmdir mlx4_0
```

(*) Possible value are "IB/RoCE V1" and "RoCE V2"

3.1.7.2.4 GID Table Example

The following is an example of the GID table.

```
DEV     PORT    INDEX   GID                                     IPv4            Type          Netdev
---     ----    -----   ---                                     ------------    ----          ------
mlx4_0  1       0       fe80:0000:0000:0000:0202:c9ff:feb6:7c70                  RoCE V2         eth1
mlx4_0  1       1       fe80:0000:0000:0000:0202:c9ff:feb6:7c70                  RoCE V1         eth1
mlx4_0  1       2       0000:0000:0000:0000:0000:ffff:c0a8:0146 192.168.1.70    RoCE V2         eth1
mlx4_0  1       3       0000:0000:0000:0000:0000:ffff:c0a8:0146 192.168.1.70    RoCE V1         eth1
mlx4_0  1      4    0000:0000:0000:0000:0000:ffff:c1a8:0146 193.168.1.70    RoCE V2        eth1.100
mlx4_0  1      5    0000:0000:0000:0000:0000:ffff:c1a8:0146 193.168.1.70    RoCE V1        eth1.100
mlx4_0  1       6       1234:0000:0000:0000:0000:0000:0000:0070                  RoCE V2         eth1
mlx4_0  1       7       1234:0000:0000:0000:0000:0000:0000:0070                  RoCE V1         eth1
mlx4_0  2       0       fe80:0000:0000:0000:0202:c9ff:feb6:7c71                  RoCE V2         eth2
mlx4_0  2       1       fe80:0000:0000:0000:0202:c9ff:feb6:7c71                  RoCE V1         eth2
```

Where:

- Entries on port 1 index 0/1 are the default GIDs, one for each supported RoCE type
- Entries on port 1 index 2/3 belong to IP address 192.168.1.70 on eth1.

- Entries on port 1 index 4/5 belong to IP address 193.168.1.70 on eth1.100.
- Packets from a QP that is associated with these GID indexes will have a VLAN header (VID=100)
- Entries on port 1 index 6/7 are IPv6 GID. Packets from a QP that is associated with these GID indexes will have an IPv6 header

3.1.7.3 RoCE Lossless Ethernet Configuration

In order to function reliably, RoCE requires a form of flow control. While it is possible to use global flow control, this is normally undesirable, for performance reasons.

The normal and optimal way to use RoCE is to use Priority Flow Control (PFC). To use PFC, it must be enabled on all endpoints and switches in the flow path.

For further information, please refer to *RoCE Over L2 Network Enabled with PFC* User Guide:

http://www.mellanox.com/related-docs/prod_software/RoCE_with_Priority_Flow_Control_Application_Guide.pdf

3.1.7.3.1 Prerequisites

The following are the driver's prerequisites in order to set or configure RoCE:

- ConnectX®-3 firmware version 2.32.5000 or higher
- ConnectX®-3 Pro firmware version 2.32.5000 or higher
- All InfiniBand verbs applications which run over InfiniBand verbs should work on RoCE links if they use GRH headers.
- Set HCA to use Ethernet protocol:
  Display the Device Manager and expand "System Devices". Please refer to Section 3.1.1.1, "ConnectX-3/ConnectX-3 Pro Port Type Management", on page 41.

3.1.7.3.2 Configuring SwitchX® Based Switch System

➢ ***To enable RoCE, the SwitchX should be configured as follows:***

- Ports facing the host should be configured as access ports, and either use global pause or Port Control Protocol (PCP) for priority flow control
- Ports facing the network should be configured as trunk ports, and use Port Control Protocol (PCP) for priority flow control

For further information on how to configure SwitchX, please refer to SwitchX User Manual.

3.1.7.3.3 Configuring DAPL over RoCE

The default dat.conf file which contains entries for the DAPL devices, does not contain entries for the DAPL over RDMA_CM over RoCE devices.

➢ ***To add the missing entries:***

Step 1. Run the ibdev2netdev utility to see all the associations between the Ethernet devices and the IB devices/ports.

Step 2. Add a new entry line according to the format below to the dat.conf file for each output line of the ibdev2netdev utility.

<IA Name> u2.0 nonthreadsafe default libdaplofa.so.2 dapl.2.0 "<ethX> <port>" ""

Parameter	Description	Example
<IA Name>	The device's IA name. The name must be unique.	ofa-v2-ethx
<ethX>	The associated Ethernet device used by RoCE.	eth3
<port>	The port number.	1

The following is an example of the ibdev2netdev utility's output and the entries added per each output line:

```
sw419:~ # ibdev2netdev mlx4_0 port 2 <===> eth2 mlx4_0 port 1 <===> eth3
ofa-v2-eth2 u2.0 nonthreadsafe default libdaplofa.so.2 dapl.2.0 "eth2 2" ""
ofa-v2-eth3 u2.0 nonthreadsafe default libdaplofa.so.2 dapl.2.0 "eth3 1" ""
```

The next section provides information of how to use InfiniBand over Ethernet (RoCE).

3.1.7.3.4 Installing and Loading the Driver

➢ ***To install and load the driver:***

Step 1. Install MLNX_OFED (See section Section 2.3, “Installing Mellanox OFED”, on page 29 for details on installation.)

RoCE is installed as part of mlx4 and mlx4_en and other modules upon driver’s installation.

The list of the modules that will be loaded automatically upon boot can be found in the configuration file /etc/infiniband/openib.conf.

Step 2. Display the existing MLNX_OFED version.

```
# ibv_devinfo
hca_id: mlx4_0
        transport: InfiniBand (0)
        fw_ver: 2.31.5050
        node_guid: 0002:c903:0008:e810
        sys_image_guid: 0002:c903:0008:e813
        vendor_id: 0x02c9
        vendor_part_id: 4099
        hw_ver: 0xB0
        board_id: MT_1060110019
        phys_port_cnt: 2
             port: 1
                   state: PORT_INIT (2)
                   max_mtu: 2048 (4)
                   active_mtu: 2048 (4)
                   sm_lid: 0
                   port_lid: 0
                   port_lmc: 0x00
                   link_layer: IB
             port: 2
                   state: PORT_ACTIVE (4)
```

```
                    max_mtu: 2048 (4)
                    active_mtu: 1024 (3)
                    sm_lid: 0
                    port_lid: 0
                    port_lmc: 0x00
                    link_layer: Ethernet
#
```

Notes about the output above:

The ports' states are:
• Port 1 - InfiniBand, is in PORT_INIT state
• Port 2 - Ethernet is in PORT_ACTIVE state
	The port state can also be obtained by running the following commands:
	`# cat /sys/class/infiniband/mlx4_0/ports/1/state 2: INIT`
	`# cat /sys/class/infiniband/mlx4_0/ports/2/state 4: ACTIVE`
	`#`

The link_layer parameter shows that:
• Port 1 is InfiniBand
• Port 2 is Ethernet
Nevertheless, Port 2 appears in the list of the HCA's ports.
	The link_layer of the two ports can also be obtained by running the following commands:
	`# cat /sys/class/infiniband/mlx4_0/ports/1/link_layer InfiniBand`
	`# cat /sys/class/infiniband/mlx4_0/ports/2/link_layer Ethernet`
	`#`

The fw_ver parameter shows that the firmware version is 2.31.5050.	The firmware version can also be obtained by running the following commands:
	`# cat /sys/class/infiniband/mlx4_0/fw_ver 2.31.5050`
	`#`

Although the InfiniBand over Ethernet's Port MTU is 2K byte at maximum, the actual MTU cannot exceed the mlx4_en interface's MTU. Since the mlx4_en interface's MTU is typically 1560, port 2 will run with MTU of 1K. Please note that RoCE's MTU are subject to InfiniBand MTU restrictions. The RoCE's MTU values are, 256 byte, 512 byte, 1024 byte and 2K. In general RoCE MTU is the largest power of 2 that is still lower than mlx4_en interface MTU.	-

3.1.7.3.5 Associating InfiniBand Ports to Ethernet Ports

Since both RoCE and mlx4_en use the Ethernet port of the adapter, one of the drivers must control the port state. In the example above, the mlx4_en driver controls the port's state. The mlx-4_ib driver holds a reference to the mlx4_en net device for getting notifications about the state of the port, as well as using the mlx4_en driver to resolve IP addresses to MAC that are required for address vector creation. However, RoCE traffic does not go through the mlx4_en driver, it is completely offloaded by the hardware.

```
# ibdev2netdev
mlx4_0 port 2 <===> eth2
mlx4_0 port 1 <===> ib0
#
```

3.1.7.3.6 Configuring an IP Address to the mlx4_en Interface

➢ ***To configure an IP address to the mlx4_en interface:***

Step 1. Configure an IP address to the mlx4_en interface on both sides of the link.

```
# ifconfig eth2 20.4.3.220
# ifconfig eth2
eth2            Link encap:Ethernet HWaddr 00:02:C9:08:E8:11
                inet addr:20.4.3.220 Bcast:20.255.255.255 Mask:255.0.0.0
                UP BROADCAST MULTICAST MTU:1500 Metric:1
                RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:1000
                RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Step 2. Make sure that ping is working.

```
# ping 20.4.3.219
PING 20.4.3.219 (20.4.3.219) 56(84) bytes of data.
64 bytes from 20.4.3.219: icmp_seq=1 ttl=64 time=0.873 ms
64 bytes from 20.4.3.219: icmp_seq=2 ttl=64 time=0.198 ms
64 bytes from 20.4.3.219: icmp_seq=3 ttl=64 time=0.167 ms
--- 20.4.3.219 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.167/0.412/0.873/0.326 ms
```

3.1.7.3.7 Adding VLANs

➢ ***To add VLANs:***

Step 1. Make sure that the 8021.q module is loaded.

```
# modprobe 8021q
```

Step 2. Add VLAN.

```
# vconfig add eth2 7
Added VLAN with VID == 7 to IF -:eth2:-
#
```

Step 3. Configure an IP address.

```
# ifconfig eth2.7 7.4.3.220
```

3.1.7.3.8 Defining Ethernet Priority (PCP in 802.1q Headers)

Step 1. Define ethernet priority on the server.

```
# ibv_rc_pingpong -g 1 -i 2 -l 4
local address: LID 0x0000, QPN 0x1c004f, PSN 0x9daf6c, GID fe80::202:c900:708:e799
remote address: LID 0x0000, QPN 0x1c004f, PSN 0xb0a49b, GID fe80::202:c900:708:e811
8192000 bytes in 0.01 seconds = 4840.89 Mbit/sec
1000 iters in 0.01 seconds = 13.54 usec/iter
```

Step 2. Define ethernet priority on the client.

```
# ibv_rc_pingpong -g 1 -i 2 -l 4 sw419
local address: LID 0x0000, QPN 0x1c004f, PSN 0xb0a49b, GID fe80::202:c900:708:e811
remote address: LID 0x0000, QPN 0x1c004f, PSN 0x9daf6c, GID fe80::202:c900:708:e799
8192000 bytes in 0.01 seconds = 4855.96 Mbit/sec
1000 iters in 0.01 seconds = 13.50 usec/iter
```

3.1.7.3.9 Using rdma_cm Tests

Step 1. Use rdma_cm test on the server.

```
# ucmatose
cmatose: starting server
initiating data transfers
completing sends
receiving data transfers
data transfers complete
cmatose: disconnecting
disconnected
test complete
return status 0
#
```

Step 2. Use rdma_cm test on the client.

```
# ucmatose -s 20.4.3.219
cmatose: starting client
cmatose: connecting
receiving data transfers
sending replies
data transfers complete
test complete
return status 0
#
```

This server-client run is without PCP or VLAN because the IP address used does not belong to a VLAN interface. If you specify a VLAN IP address, then the traffic should go over VLAN.

3.1.7.4 Type Of Service (TOS)

The TOS field for rdma_cm sockets can be set using the rdma_set_option() API, just as it is set for regular sockets. If a TOS is not set, the default value (0) is used. Within the rdma_cm kernel driver, the TOS field is converted into an SL field. The conversion formula is as follows:

- SL = TOS >> 5 (e.g., take the 3 most significant bits of the TOS field)

In the hardware driver, the SL field is converted into PCP by the following formula:

- PCP = SL & 7 (take the 3 least significant bits of the TOS field)

SL affects the PCP only when the traffic goes over tagged VLAN frames.

3.1.7.5 RoCE Link Aggregation (RoCE LAG)

Supported in ConnectX-3 and ConnectX-3 Pro only.

RoCE Link Aggregation provides failover and link aggregation capabilities for mlx4 device physical ports. In this mode, only one IB port, that represents the two physical ports, is exposed to the application layer.

3.1.7.5.1 Enabling RoCE Link Aggregation

To enter the Link Aggregation mode, a bonding master that enslaves the two net devices on the mlx4 ports is required. Then, the mlx4 device re-registers itself in the IB stack with a single port. If the requirement is not met, the device re-registers itself again with two ports.

For the device to enter the Link Aggregation mode, the following prerequisites must exist:

- Exactly 2 slaves must be under the bonding master
- The bonding master has to be in one of the following modes:
  - (1) active-backup mode
  - (2) static active-active mode
  - (4) dynamic active-active mode

Restarting the device, when entering or leaving Link Aggregation mode, invalidates the open resources (QPs, MRs, etc.) on the device.

3.1.7.5.1.1 Link Aggregation in active-backup Mode

When the bonding master works in active-backup mode, RoCE packets are transmitted and received from the active port that the bonding master reports. The logic of fail over is done solely in the bonding driver and the mlx4 driver only polls it.

3.1.7.5.1.2 Link Aggregation in active-active Mode

In this mode, RoCE packets are transmitted and received from both physical ports. While the mlx4 driver has no influence on the port on which packets are being received from, it can determine the port packets are transmitted to.

If user application does not set a preference, the mlx4 driver chooses a port in a round robin fashion when QP is modified from RESET to INIT. This is necessary because application sees only one port to use so it will always state `port_num` 1 in the QP attributes. With that, the theoretical bandwidth of the system will be kept as the sum of the two ports.

Application that prefers to send packets on the specific port for a specific QP, should set `flow_entropy` when modifying a QP from RESET to INIT. Values for the `flow_entropy` parameter are interpreted by the mlx4 driver as a hint to associate the SQ of the QP to "`1`" while odd values associate the SQ with port 2.

The code example below shows how to set `flow_entropy` for a QP.

```
struct ibv_exp_qp_attr attr = {
                      .comp_mask       = IBV_EXP_QP_ATTR_FLOW_ENTROPY,
                      .qp_state        = IBV_QPS_INIT,
                      .pkey_index      = 0,
                      .port_num        = port,
                      .qp_access_flags = 0,
                      .flow_entropy    = 1
              };

              if (ibv_exp_modify_qp(ctx->qp, &attr,
                                IBV_QP_STATE              |
                                IBV_QP_PKEY_INDEX         |
                                IBV_QP_PORT               |
                                IBV_EXP_QP_FLOW_ENTROPY   |
                                IBV_QP_ACCESS_FLAGS)) {
                      fprintf(stderr, "Failed to modify QP to INIT\n");
                      goto clean_qp;
              }
```

3.1.8 Explicit Congestion Notification (ECN)

3.1.8.1 ConnectX-3/ConnectX-3 Pro ECN

ECN is an extension to the IP protocol. It allows reliable communication by notifying all ends of communication when a congestion occurs.

This is done without dropping packets. Please note that since this feature requires all nodes in the path (nodes, routers etc) between the communicating nodes to support ECN to ensure reliable communication. ECN is marked as 2 bits in the traffic control IP header. This ECN implementation refers to RoCE v2.

ECN command interface is use to configure ECN activity. The access to it is through the file system (mount of debugfs is required). The interface provides a set of changeable attributes, and information regarding ECN's counters and statistics.

Enabling the ECN command interface is done by setting the `en_ecn` module parameter of mlx-4_ib to 1:

```
options mlx4_ib en_ecn=1
```

3.1.8.1.1 Enabling ECN

➢ ***To enable ECN on the hosts***

Step 1. Enable ECN in sysfs.

```
/proc/sys/net/ipv4/tcp_ecn = 1
```

Step 2. Enable ECN CLI.

```
options mlx4_ib en_ecn=1
```

Step 3. Restart the driver.

```
/etc/init.d/openibd restart
```

Step 4. Mount debugfs to access ECN attributes.

```
 mount -t debugfs none /sys/kernel/debug/
```

Please note, mounting of debugfs is required.

The following is an example for ECN configuration through debugfs (echo 1 to enable attribute):

```
/sys/kernel/debug/mlx4_ib/<device>/ecn/<algo>/ports/1/params/prios/<prio>/<the
requested attribute>
```

ECN supports the following algorithms:

- `r_roce_ecn_rp`
- `r_roce_ecn_np`

Each algorithm has a set of relevant parameters and statistics, which are defined per device, per port, per priority.

`r_roce_ecn_np` has an extra set of general parameters which are defined per device.

ECN and QCN are not compatible. When using ECN, QCN (and all its related daemons/utilities that could enable it, i.e - lldpad) should be turned OFF.

3.1.8.1.2 Various ECN Paths

The following are the paths to ECM algorithm, general parameters and counters.

- The path to an algorithm attribute is (except for general parameters):

```
/sys/kernel/debug/mlx4_ib/{DEVICE}/ ecn/{algorithm}/ports/{port}/params/prios/{prio}/
{attribute}
```

- The path to a general parameter is:

```
/sys/kernel/debug/mlx4_ib/{DEVICE}/ ecn/r_roce_ecn_np/gen_params/{attribute}
```

- The path to a counter is:

```
/sys/kernel/debug/mlx4_ib/{DEVICE}/ ecn/{algorithm}/ports/{port}/statistics/prios/
{prio}/{counter}
```

3.1.8.2 ConnectX-4 ECN

ECN in ConnectX-4 enables end-to-end congestions notifications between two end-points when a congestion occurs, and works over Layer 3. ECN must be enabled on all nodes in the path (nodes, routers etc) between the two end points and the intermediate devices (switches) between them to ensure reliable communication.

3.1.8.2.1 Enabling ECN

➢ ***To enable ECN on the hosts***

Step 1. Enable ECN in sysfs.

```
/sys/class/net/<interface>/<protocol>/ecn_<protocol>_enable =1
```

Step 2. Query the attribute.

```
cat /sys/class/net/<interface>/ecn/<protocol>/params/<requested attribute>
```

Step 3. Modify the attribute.

```
echo <value> /sys/class/net/<interface>/ecn/<protocol>/params/<requested attribute>
```

ECN supports the following algorithms:

- `r_roce_ecn_rp - Reaction point`

- `r_roce_ecn_np - Notification point`

Each algorithm has a set of relevant parameters and statistics, which are defined per device, per port, per priority.

➢ ***To query ECN enable per Priority X:***

```
cat /sys/class/net/<interface>/ecn/<protocol>/enable/X
```

➢ ***To read ECN configurable parameters:***

```
cat  /sys/class/net/<interface>/ecn/<protocol>/requested attributes
```

➢ ***To enabled ECN for each priority per protocol:***

```
echo 1 >  /sys/class/net/<interface>/ecn/<protocol>/enable/X
```

➢ ***To modify ECN configurable parameters:***

```
echo <value> >  /sys/class/net/<interface>/ecn/<protocol>/requested attributes
```

Where:

- X: priority {0..7}
- protocol: roce_rp / roce_np
- requested attributes: Next Slide for each protocol.

3.1.9 RSS Support

3.1.9.1 RSS Hash Function

The device has the ability to use XOR as the RSS distribution function, instead of the default Toplitz function.

The XOR function can be better distributed among driver's receive queues in small number of streams, where it distributes each TCP/UDP stream to a different queue.

MLNX_OFED provides an option to change the working RSS hash function from Toplitz to XOR (and vice versa) through ethtool priv-flags.

For further information, please refer to Table 1, “ethtool Supported Options,” on page 57.

3.1.9.1.1 RSS Support for IP Fragments

As of MLNX_OFED v2.4-.1.0.0, RSS will distribute incoming IP fragmented datagrams according to its hash function, considering the L3 IP header values. Different IP fragmented datagrams flows will be directed to different rings.

When the first packet in IP fragments chain contains upper layer transport header (e.g. UDP packets larger than MTU), it will be directed to the same target as the proceeding IP fragments that follows it, to prevent out-of-order processing.

3.1.9.2 RSS Verbs Support for ConnectX-4 HCAs

Receive Side Scaling (RSS) technology allows spreading incoming traffic between different receive descriptor queues. Assigning each queue to different CPU cores allows better load balancing of the incoming traffic and improve performance.

This technology was extend to user space by the verbs layer and can be used for RAW ETH QP.

3.1.9.2.1 RSS Flow Steering

Steering rules classify incoming packets and deliver a specific traffic type (e.g. TCP/UDP, IP only) or a specific flow to "RX Hash" QP. "RX Hash" QP is responsible for spreading the traffic it handles between the Receive Work Queues using RX hash and Indirection Table. The Receive Work Queue can point to different CQs that can be associated with different CPU cores.

3.1.9.2.2 Verbs

The below experimental verbs should be used to achieve this task in both control and data path.

Details per verb should be referenced from its man page.

- Control path:
  - ibv_exp_create_wq, ibv_exp_modify_wq, ibv_exp_destory_wq
  - ibv_exp_create_rwq_ind_table, ibv_exp_destroy_rwq_ind_table
  - ibv_exp_create_qp with specific RX configuration to create the "RX hash" QP.
  - ibv_exp_query_device
- Data path

  The accelerated verbs should be used to post receive to the created WQ(s) and to poll for completions. Specifically, ibv_exp_query_intf should be used to get IBV_EXP_INTF_WQ family and work with its functions to post receive.

  For the polling should use the IBV_EXP_INTF_CQ family with poll_length which fits the receive side.

3.1.10 Time-Stamping

Supported in ConnectX-3 and ConnectX-3 Pro only.

3.1.10.1 Time-Stamping Service

Time stamping is the process of keeping track of the creation of a packet. A time-stamping service supports assertions of proof that a datum existed before a particular time. Incoming packets are time-stamped before they are distributed on the PCI depending on the congestion in the PCI buffers. Outgoing packets are time-stamped very close to placing them on the wire.

Enabling Time Stamping

Time-stamping is off by default and should be enabled before use.

➢ ***To enable time stamping for a socket:***

- Call setsockopt() with SO_TIMESTAMPING and with the following flags:

```
SOF_TIMESTAMPING_TX_HARDWARE:  try to obtain send time stamp in hardware
SOF_TIMESTAMPING_TX_SOFTWARE:  if SOF_TIMESTAMPING_TX_HARDWARE is off or
                               fails, then do it in software
SOF_TIMESTAMPING_RX_HARDWARE:  return the original, unmodified time stamp
                               as generated by the hardware
SOF_TIMESTAMPING_RX_SOFTWARE:  if SOF_TIMESTAMPING_RX_HARDWARE is off or
                               fails, then do it in software
```

```
SOF_TIMESTAMPING_RAW_HARDWARE: return original raw hardware time stamp
SOF_TIMESTAMPING_SYS_HARDWARE: return hardware time stamp transformed to
                               the system time base
SOF_TIMESTAMPING_SOFTWARE:     return system time stamp generated in
                               software
SOF_TIMESTAMPING_TX/RX determine how time stamps are generated.
SOF_TIMESTAMPING_RAW/SYS determine how they are reported
```

➢ ***To enable time stamping for a net device:***

Admin privileged user can enable/disable time stamping through calling ioctl (sock, SIOCSH-WTSTAMP, &ifreq) with following values:

Send side time sampling:

- Enabled by ifreq.hwtstamp_config.tx_type when

```
/* possible values for hwtstamp_config->tx_type */
enum hwtstamp_tx_types {
        /*
         * No outgoing packet will need hardware time stamping;
         * should a packet arrive which asks for it, no hardware
         * time stamping will be done.
        */
        HWTSTAMP_TX_OFF,

        /*
         * Enables hardware time stamping for outgoing packets;
         * the sender of the packet decides which are to be
         * time stamped by setting %SOF_TIMESTAMPING_TX_SOFTWARE
         * before sending the packet.
         */
        HWTSTAMP_TX_ON,
  /*
         * Enables time stamping for outgoing packets just as
         * HWTSTAMP_TX_ON does, but also enables time stamp insertion
         * directly into Sync packets. In this case, transmitted Sync
         * packets will not received a time stamp via the socket error
         * queue.
         */
        HWTSTAMP_TX_ONESTEP_SYNC,
};
Note: for send side time stamping currently only HWTSTAMP_TX_OFF and
HWTSTAMP_TX_ON are supported.
```

Receive side time sampling:

- Enabled by ifreq.hwtstamp_config.rx_filter when

```
/* possible values for hwtstamp_config->rx_filter */
enum hwtstamp_rx_filters {
        /* time stamp no incoming packet at all */
        HWTSTAMP_FILTER_NONE,

        /* time stamp any incoming packet */
        HWTSTAMP_FILTER_ALL,
      /* return value: time stamp all packets requested plus some others */
        HWTSTAMP_FILTER_SOME,

        /* PTP v1, UDP, any kind of event packet */
        HWTSTAMP_FILTER_PTP_V1_L4_EVENT,
        /* PTP v1, UDP, Sync packet */
        HWTSTAMP_FILTER_PTP_V1_L4_SYNC,
        /* PTP v1, UDP, Delay_req packet */
        HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ,
        /* PTP v2, UDP, any kind of event packet */
        HWTSTAMP_FILTER_PTP_V2_L4_EVENT,
        /* PTP v2, UDP, Sync packet */
        HWTSTAMP_FILTER_PTP_V2_L4_SYNC,
        /* PTP v2, UDP, Delay_req packet */
        HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ,

        /* 802.AS1, Ethernet, any kind of event packet */
        HWTSTAMP_FILTER_PTP_V2_L2_EVENT,
        /* 802.AS1, Ethernet, Sync packet */
        HWTSTAMP_FILTER_PTP_V2_L2_SYNC,
        /* 802.AS1, Ethernet, Delay_req packet */
        HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ,

        /* PTP v2/802.AS1, any layer, any kind of event packet */
        HWTSTAMP_FILTER_PTP_V2_EVENT,
        /* PTP v2/802.AS1, any layer, Sync packet */
        HWTSTAMP_FILTER_PTP_V2_SYNC,
        /* PTP v2/802.AS1, any layer, Delay_req packet */
        HWTSTAMP_FILTER_PTP_V2_DELAY_REQ,
};
Note: for receive side time stamping currently only HWTSTAMP_FILTER_NONE and
HWTSTAMP_FILTER_ALL are supported.
```

3.1.10.1.1Getting Time Stamping

Once time stamping is enabled time stamp is placed in the socket Ancillary data. recvmsg() can be used to get this control message for regular incoming packets. For send time stamps the outgoing packet is looped back to the socket's error queue with the send time stamp(s) attached. It can be received with recvmsg(flags=MSG_ERRQUEUE). The call returns the original outgoing packet data including all headers preprended down to and including the link layer, the scm_timestamping control message and a sock_extended_err control message with ee_errno==ENOMSG and ee_origin==SO_EE_ORIGIN_TIMESTAMPING. A socket with such a pending bounced packet is ready for reading as far as select() is concerned. If the outgoing packet has to be fragmented, then only the first fragment is time stamped and returned to the sending socket.

When time-stamping is enabled, VLAN stripping is disabled. For more info please refer to Documentation/networking/timestamping.txt in kernel.org

3.1.10.1.2Querying Time Stamping Capabilities via ethtool

➢ ***To display Time Stamping capabilities via ethtool:***

- Show Time Stamping capabilities

```
ethtool -T eth<x>
```

Example:

```
ethtool -T eth0
Time stamping parameters for p2p1:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

For more details on PTP Hardware Clock, please refer to:

https://www.kernel.org/doc/Documentation/ptp/ptp.txt

3.1.10.2 RoCE Time Stamping

RoCE Time Stamping is currently at beta level.
Please be aware that everything listed here is subject to change.

Supported in ConnectX-3 and ConnectX-3 Pro only.

RoCE Time Stamping allows you to stamp packets when they are sent to the wire / received from the wire. The time stamp is given in a raw hardware cycles, but could be easily converted into hardware referenced nanoseconds based time. Additionally, it enables you to query the hardware for the hardware time, thus stamp other application's event and compare time.

3.1.10.2.1Query Capabilities

Time stamping is available if and only the hardware reports it is capable of reporting it. To verify whether RoCE Time Stamping is available, run `ibv_exp_query_device`.

For example:

```
struct ibv_exp_device_attr attr;
ibv_exp_query_device(context, &attr);
if (attr.comp_mask & IBV_EXP_DEVICE_ATTR_WITH_TIMESTAMP_MASK) {
if (attr.timestamp_mask) {
                /* Time stamping is supported with mask attr.timestamp_mask */
        }
}
if (attr.comp_mask & IBV_EXP_DEVICE_ATTR_WITH_HCA_CORE_CLOCK) {
        if (attr.hca_core_clock) {
                /* reporting the device's clock is supported. */
                /* attr.hca_core_clock is the frequency in MHZ */
        }
}
```

3.1.10.2.2Creating Time Stamping Completion Queue

To get time stamps, a suitable extended Completion Queue (CQ) must be created via a special call to `ibv_exp_create_cq` verb.

```
cq_init_attr.flags = IBV_EXP_CQ_TIMESTAMP;
cq_init_attr.comp_mask = IBV_EXP_CQ_INIT_ATTR_FLAGS;
cq = ibv_exp_create_cq(context, cqe, node, NULL, 0, &cq_init_attr);
```

This CQ cannot report SL or SLID information. The value of `sl` and `sl_id` fields in `struct ibv_exp_wc` are invalid. Only the fields indicated by the `exp_wc_flags` field in `struct ibv_exp_wc` contains a valid and usable value.

When using Time Stamping, several fields of `struct ibv_exp_wc` are not available resulting in RoCE UD / RoCE traffic with VLANs failure.

3.1.10.2.3Polling a Completion Queue

Polling a CQ for time stamp is done via the `ibv_exp_poll_cq` verb.

```
ret = ibv_exp_poll_cq(cq, 1, &wc_ex, sizeof(wc_ex));
if (ret > 0) {
                /* CQ returned a wc */
                if (wc_ex.exp_wc_flags & IBV_EXP_WC_WITH_TIMESTAMP) {
                /* This wc contains a timestamp */
                timestamp = wc_ex.timestamp;
                /* Timestamp is given in raw hardware time */
                }
}
```

CQs that are opened with the `ibv_exp_create_cq` verbs should be always be polled with the `ibv_exp_poll_cq` verb.

3.1.10.2.4Querying the Hardware Time

Querying the hardware for time is done via the `ibv_exp_query_values` verb.

For example:

```
ret = ibv_exp_query_values(context, IBV_EXP_VALUES_HW_CLOCK, &queried_values);
if (!ret && queried_values.comp_mask & IBV_EXP_VALUES_HW_CLOCK)
queried_time = queried_values.hwclock;
```

To change the queried time in nanoseconds resolution, use the `IBV_EXP_VALUES_HW_CLOCK_NS` flag along with the `hwclock_ns` field.

```
ret = ibv_exp_query_values(context, IBV_EXP_VALUES_HW_CLOCK_NS, &queried_values);
if (!ret && queried_values.comp_mask & IBV_EXP_VALUES_HW_CLOCK_NS)
queried_time_ns = queried_values.hwclock_ns;
```

Querying the Hardware Time is available only on physical functions / native machines.

3.1.11 Flow Steering

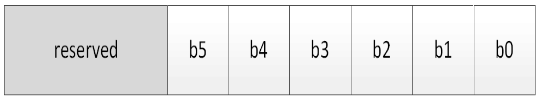
The mlx5 driver supports only Ethernet L2 Flow Steering.

Flow steering is a new model which steers network flows based on flow specifications to specific QPs. Those flows can be either unicast or multicast network flows. In order to maintain flexibility, domains and priorities are used. Flow steering uses a methodology of flow attribute, which is a combination of L2-L4 flow specifications, a destination QP and a priority. Flow steering rules may be inserted either by using ethtool or by using InfiniBand verbs. The verbs abstraction uses a different terminology from the flow attribute (`ibv_exp_flow_attr`), defined by a combination of specifications (`struct ibv_exp_flow_spec_*`).

3.1.11.1 Enable/Disable Flow Steering

Only applicable to the mlx4 driver. Flow Steering is automatically enabled in the mlx5 driver as of MLNX_OFED v3.1-x.0.0 and above.

Flow steering is generally enabled when the `log_num_mgm_entry_size` module parameter is non positive (e.g., `-log_num_mgm_entry_size`), meaning the absolute value of the parameter, is a bit field. Every bit indicates a condition or an option regarding the flow steering mechanism:



reserved	b5	b4	b3	b2	b1	b0

bit	Operation	Description
b0	Force device managed Flow Steering	When set to 1, it forces HCA to be enabled regardless of whether NC-SI Flow Steering is supported or not.
b1	Disable IPoIB Flow Steering	When set to 1, it disables the support of IPoIB Flow Steering. This bit should be set to 1 when "b2- Enable A0 static DMFS steering" is used (see Section 3.1.11.3, “A0 Static Device Managed Flow Steering”, on page 81).
b2	Enable A0 static DMFS steering (see Section 3.1.11.3, “A0 Static Device Managed Flow Steering”, on page 81)	When set to 1, A0 static DMFS steering is enabled. This bit should be set to 0 when "b1- Disable IPoIB Flow Steering" is 0.
b3	Enable DMFS only if the HCA supports more than 64QPs per MCG entry	When set to 1, DMFS is enabled only if the HCA supports more than 64 QPs attached to the same rule. For example, attaching 64VFs to the same multicast address causes 64QPs to be attached to the same MCG. If the HCA supports less than 64 QPs per MCG, B0 is used.
b4	Optimize IPoIB/EoIB steering table for non source IP rules when possible	When set to 1, IPoIB/EoIB steering table will be optimized to support rules ignoring source IP check. This optimization is available only when IPoIB Flow Steering is set.
b5	Optimize steering table for non source IP rules when possible	When set to 1, steering table will be optimized to support rules ignoring source IP check. This optimization is possible only when DMFS mode is set.

For example, a value of (-7) means:

- forcing Flow Steering regardless of NC-SI Flow Steering support
- disabling IPoIB Flow Steering support
- enabling A0 static DMFS steering
- steering table is not optimized for rules ignoring source IP check

The default value of `log_num_mgm_entry_size` is -10. Meaning Ethernet Flow Steering (i.e IPoIB DMFS is disabled by default) is enabled by default if NC-SI DMFS is supported and the HCA supports at least 64 QPs per MCG entry. Otherwise, L2 steering (B0) is used.

When using SR-IOV, flow steering is enabled if there is an adequate amount of space to store the flow steering table for the guest/master.

➤ ***To enable Flow Steering:***

Step 1. Open the `/etc/modprobe.d/mlnx.conf` file.

Step 2. Set the parameter `log_num_mgm_entry_size` to a non positive value by writing the option `mlx4_core log_num_mgm_entry_size=<value>`.

Step 3. Restart the driver

➤ ***To disable Flow Steering:***

Step 1. Open the `/etc/modprobe.d/mlnx.conf` file.

Step 2. Remove the `options mlx4_core log_num_mgm_entry_size= <value>`.

Step 3. Restart the driver

3.1.11.2 Flow Steering Support

Only applicable to the mlx4 driver.

➤ ***To determine which Flow Steering features are supported:***

```
ethtool --show-priv-flags eth4
```

The following output is shown:

```
mlx4_flow_steering_ethernet_l2: on      Creating Ethernet L2 (MAC) rules is supported
mlx4_flow_steering_ipv4: on             Creating IPv4 rules is supported
mlx4_flow_steering_tcp: on              Creating TCP/UDP rules is supported
```

Flow Steering support in InfiniBand is determined according to the `EXP_MANAGED_-FLOW_STEERING` flag.

3.1.11.3 A0 Static Device Managed Flow Steering

Only applicable to the mlx4 driver.

This mode enables fast steering, however it might impact flexibility. Using it increases the packet rate performance by ~30%, with the following limitations for Ethernet link-layer unicast QPs:

- Limits the number of opened RSS Kernel QPs to 96. MACs should be unique (1 MAC per 1 QP). The number of VFs is limited.

- When creating Flow Steering rules for user QPs, only MAC--> QP rules are allowed. Both MACs and QPs should be unique between rules. Only 62 such rules could be created
- When creating rules with Ethtool, MAC--> QP rules could be used, where the QP must be the indirection (RSS) QP. Creating rules that indirect traffic to other rings is not allowed. Ethtool MAC rules to drop packets (action -1) are supported.
- RFS is not supported in this mode
- VLAN is not supported in this mode

3.1.11.4 Flow Domains and Priorities

The mlx5 driver supports only User Verbs domain with struct ibv_exp_flow_spec_eth flow specification using 4 priorities.

Flow steering defines the concept of domain and priority. Each domain represents a user agent that can attach a flow. The domains are prioritized. A higher priority domain will always supersede a lower priority domain when their flow specifications overlap. Setting a lower priority value will result in higher priority.

In addition to the domain, there is priority within each of the domains. Each domain can have at most 2^12 priorities in accordance with its needs.

The following are the domains at a descending order of priority:

- **User Verbs** allows a user application QP to be attached into a specified flow when using `ibv_exp_create_flow` and `ibv_exp_destroy_flow` verbs
  - `ibv_exp_create_flow`

```
struct ibv_exp_flow *ibv_exp_create_flow(struct ibv_qp *qp, struct ibv_exp_flow_attr
*flow)
```

Input parameters:

- `struct ibv_qp` - the attached QP.
- `struct ibv_exp_flow_attr` - attaches the QP to the flow specified. The flow contains mandatory control parameters and optional L2, L3 and L4 headers. The optional headers are detected by setting the size and `num_of_specs` fields:

  `struct ibv_exp_flow_attr` can be followed by the optional flow headers structs:

```
struct ibv_exp_flow_spec_ib
struct ibv_exp_flow_spec_eth
struct ibv_exp_flow_spec_ipv4
struct ibv_exp_flow_spec_tcp_udp
```

For further information, please refer to the `ibv_exp_create_flow` man page.

Be advised that as of MLNX_OFED v2.0-3.0.0, the parameters (both the value and the mask) should be set in big-endian format.

Each header struct holds the relevant network layer parameters for matching. To enforce the match, the user sets a mask for each parameter.

The mlx5 driver supports partial masks. The mlx4 driver supports the following masks:

- All one mask - include the parameter value in the attached rule
  Note: Since the VLAN ID in the Ethernet header is 12bit long, the following parameter should be used: `flow_spec_eth.mask.vlan_tag = htons(0x0fff).`
- All zero mask - ignore the parameter value in the attached rule

When setting the flow type to NORMAL, the incoming traffic will be steered according to the rule specifications. ALL_DEFAULT and MC_DEFAULT rules options are valid only for Ethernet link type since InfiniBand link type packets always include QP number.

For further information, please refer to the relevant man pages.

- `ibv_exp_destroy_flow`

```
int ibv_exp_destroy_flow(struct ibv_exp_flow *flow_id)
```

Input parameters:

`ibv_exp_destroy_flow` requires `struct ibv_exp_flow` which is the return value of `ibv_exp_create_flow` in case of success.

Output parameters:

Returns 0 on success, or the value of errno on failure.

For further information, please refer to the `ibv_exp_destroy_flow` man page.

- **Ethtool**

Ethtool domain is used to attach an RX ring, specifically its QP to a specified flow.

Please refer to the most recent ethtool manpage for all the ways to specify a flow.

Examples:

- ethtool –U eth5 flow-type ether dst 00:11:22:33:44:55 loc 5 action 2

  All packets that contain the above destination MAC address are to be steered into rx-ring 2 (its underlying QP), with priority 5 (within the ethtool domain)
- ethtool –U eth5 flow-type tcp4 src-ip 1.2.3.4 dst-port 8888 loc 5 action 2

  All packets that contain the above destination IP address and source port are to be steered into rx-ring 2. When destination MAC is not given, the user's destination MAC is filled automatically.
- ethtool -U eth5 flow-type ether dst 00:11:22:33:44:55 vlan 45 m 0xf000 loc 5 action 2

  All packets that contain the above destination MAC address and specific VLAN are steered into ring 2. Please pay attention to the VLAN's mask 0xf000. It is required in order to add such a rule.
- ethtool –u eth5

  Shows all of ethtool's steering rule

When configuring two rules with the same priority, the second rule will overwrite the first one, so this ethtool interface is effectively a table. Inserting Flow Steering rules in the kernel requires support from both the ethtool in the user space and in kernel (v2.6.28).

MLX4 Driver Support

The mlx4 driver supports only a subset of the flow specification the ethtool API defines. Asking for an unsupported flow specification will result with an “invalid value” failure.

The following are the flow specific parameters:

	ether	tcp4/udp4	ip4
Mandatory	dst		src-ip/dst-ip

	ether	tcp4/udp4	ip4
Optional	vlan	src-ip, dst-ip, src-port, dst-port, vlan	src-ip, dst-ip, vlan

- **RFS**

  RFS is an in-kernel-logic responsible for load balancing between CPUs by attaching flows to CPUs that are used by flow's owner applications. This domain allows the RFS mechanism to use the flow steering infrastructure to support the RFS logic by implementing the `ndo_rx_-flow_steer`, which, in turn, calls the underlying flow steering mechanism with the RFS domain.

  Enabling the RFS requires enabling the `'ntuple'` flag via the ethtool,

  For example, to enable ntuple for eth0, run:

  ```
  ethtool -K eth0 ntuple on
  ```

  RFS requires the kernel to be compiled with the `CONFIG_RFS_ACCEL` option. This options is available in kernels 2.6.39 and above. Furthermore, RFS requires Device Managed Flow Steering support.

  RFS cannot function if LRO is enabled. LRO can be disabled via ethtool.

- **All of the rest**

  The lowest priority domain serves the following users:

  - **The mlx4 Ethernet driver** attaches its unicast and multicast MACs addresses to its QP using L2 flow specifications
  - **The mlx4 ipoib driver** when it attaches its QP to his configured GIDS

  Fragmented UDP traffic cannot be steered. It is treated as 'other' protocol by hardware (from the first packet) and not considered as UDP traffic.

  We recommend using `libibverbs` v2.0-3.0.0 and `libmlx4` v2.0-3.0.0 and higher as of MLNX_OFED v2.0-3.0.0 due to API changes.

3.1.12 WQE Format in MLNX_OFED

To enable WQE format, libmlx4 and Mellanox drivers should be recompiled with a special flag.

Below are the instructions for compiling and installing the user space and kernel packages with the special flag `"--with-wqe-format"`.

Step 1. Download the MLNX_OFED tarball/iso file and extract/mount it.

Step 2. Install MLNX_OFED.

Step 3. Extract the source packages from MLNX_OFED.

```
# cd /tmp
# tar xzf <path to MLNX_OFED package>/src/MLNX_OFED_SRC-3.0-1.0.0.tgz
# cd /tmp/MLNX_OFED_SRC-3.0-1.0.0
```

Step 4. Build new RPMs

Step a. Build libmlx4:

```
# LDFLAGS=' -g -O3' CFLAGS=' -g -O3' CPPFLAGS=' -g -O3' CXXFLAGS=' -g -O3'
FFLAGS=' -g -O3' LDLIBS=' -g -O3' rpmbuild --rebuild  --define "_topdir $PWD/
libmlx4" --define 'dist %{nil}' --target x86_64 --define '_prefix /usr' --
define '_exec_prefix /usr' --define '_sysconfdir /etc' --define '_usr /usr' -
-define 'configure_options --with-wqe-format ' SRPMS/libmlx4-*.src.rpm
```

The RPMs will be created under `/tmp/MLNX_OFED_SRC-3.0-1.0.0/libmlx4/RPMS/x86_64/`.

Step b. Build mlnx-ofa_kernel:

```
# rpmbuild --rebuild  --define "_topdir $PWD/kernel" --nodeps --define '_dist
%{nil}' --define 'configure_options --with-wqe-format  --with-core-mod --
with-user_mad-mod --with-user_access-mod --with-addr_trans-mod --with-mthca-
mod --with-mlx4-mod --with-mlx4_en-mod --with-mlx4_vnic-mod --with-mlx5-mod -
-with-cxgb3-mod --with-cxgb4-mod --with-nes-mod --with-qib-mod --with-ipoib-
mod --with-ipath_inf-mod --with-amso1100-mod --with-sdp-mod --with-srp-mod --
with-rds-mod --with-iser-mod --with-e_ipoib-mod --with-nfsrdma-mod --with-
9pnet_rdma-mod --with-9p-mod --with-cxgb3i-mod --with-cxgb4i-mod' --define
'_prefix /usr' SRPMS/mlnx-ofa_kernel-3.0-*.src.rpm
```

The RPMs will be created under `/tmp/MLNX_OFED_SRC-3.0-1.0.0/kernel/RPMS/x86_64/`.

Step 5. Remove old installed RPMs.

```
# rpm -e --nodeps $(rpm -qa | grep -E "mlnx-ofa_kernel|libmlx4")
```

Step 6. Install new RPMs.

```
# rpm  -ivh --nodeps --force /tmp/MLNX_OFED_SRC-3.0-1.0.0/kernel/RPMS/x86_64/kernel-ib-
* /tmp/MLNX_OFED_SRC-3.0-1.0.0/libmlx4/RPMS/x86_64/libmlx4-1* /tmp/MLNX_OFED_SRC-3.0-
1.0.0/libmlx4/RPMS/x86_64/libmlx4-devel-*
```

Step 7. Reload the driver.

```
# /etc/init.d/openibd restart
```

When WQE Format is enabled, the maximum message length of scatter gather entry is 1GB – 1.

3.1.13 Wake-on-LAN (WoL)

Wake-on-LAN (WOL) is a technology that allows a network professional to remotely power on a computer or to wake it up from sleep mode.

- To enable WoL:

```
# ethtool -s <interface> wol g
```

- To get WoL:

```
ethtool <interface> | grep Wake-on
Wake-on: g
```

Where:

"`g`" is the magic packet activity.

3.1.14 Hardware Accelerated 802.1ad VLAN (Q-in-Q Tunneling)

Q-in-Q tunneling allows the user to create a Layer 2 Ethernet connection between two servers. The user can segregate a different VLAN traffic on a link or bundle different VLANs into a single VLAN. Q-in-Q tunneling adds a service VLAN tag before the user's 802.1Q VLAN tags.

➢ ***To enable device support for accelerated 802.1ad VLAN.***

1. Turn on the new ethtool private flag "`phv-bit`" (disabled by default).

```
$ ethtool --set-priv-flags eth1 phv-bit on
```

Enabling this flag sets the phv_en port capability.

2. Change the interface device features by turning on the ethtool device feature "`tx-vlan-stag-hw-insert`" (disabled by default).

```
$ ethtool -K eth1  tx-vlan-stag-hw-insert on
```

Once the private flag and the ethtool device feature are set, the device will be ready for 802.1ad VLAN acceleration.

The "`phv-bit`" private flag setting is available for the Physical Function (PF) only.
The Virtual Function (VF) can use the VLAN acceleration by setting the "`tx-vlan-stag-hw-insert`" parameter only if the private flag "`phv-bit`" is enabled by the PF. If the PF enables/disables the "`phv-bit`" flag after the VF driver is up, the configuration will take place only after the VF driver is restarted.

3.2 InfiniBand Network

3.2.1 Interface

3.2.1.1 ConnectX-3/ConnectX-3 Pro Port Type Management

For further information, please refer to Section 3.1.1.1, "ConnectX-3/ConnectX-3 Pro Port Type Management", on page 41

3.2.1.2 ConnectX-4 Port Type Management

For further information, please refer to Section 3.1.1.2, "ConnectX-4 Port Type Management", on page 42.

3.2.2 OpenSM

OpenSM is an InfiniBand compliant Subnet Manager (SM). It is provided as a fixed flow executable called "opensm", accompanied by a testing application called "osmtest". OpenSM implements an InfiniBand compliant SM according to the InfiniBand Architecture Specification chapters: Management Model (13), Subnet Management (14), and Subnet Administration (15).

3.2.2.1 opensm

opensm is an InfiniBand compliant Subnet Manager and Subnet Administrator that runs on top of the Mellanox OFED stack. **opensm** performs the InfiniBand specification's required tasks for initializing InfiniBand hardware. One SM must be running for each InfiniBand subnet.

opensm also provides an experimental version of a performance manager.

opensm defaults were designed to meet the common case usage on clusters with up to a few hundred nodes. Thus, in this default mode, **opensm** will scan the IB fabric, initialize it, and sweep occasionally for changes.

opensm attaches to a specific IB port on the local machine and configures only the fabric connected to it. (If the local machine has other IB ports, **opensm** will ignore the fabrics connected to those other ports). If no port is specified, **opensm** will select the first "best" available port. **opensm** can also present the available ports and prompt for a port number to attach to.

By default, the **opensm** run is logged to two files: `/var/log/messages` and `/var/log/opensm.log`. The first file will register only general major events, whereas the second file will include details of reported errors. All errors reported in this second file should be treated as indicators of IB fabric health issues. (Note that when a fatal and non-recoverable error occurs, **opensm** will exit). Both log files should include the message "SUBNET UP" if **opensm** was able to setup the subnet correctly.

Syntax

```
opensm [OPTIONS]
```

For the complete list of opensm options, please run:

```
opensm --help / -h / -?
```

3.2.2.1.1 Environment Variables

The following environment variables control `opensm` behavior:

- OSM_TMP_DIR

  Controls the directory in which the temporary files generated by opensm are created. These files are: `opensm-subnet.lst`, `opensm.fdbs`, and `opensm.mcfdbs`. By default, this directory is `/var/log`.
- OSM_CACHE_DIR

  `opensm` stores certain data to the disk such that subsequent runs are consistent. The default directory used is `/var/cache/opensm`. The following file is included in it:
  - `guid2lid` – stores the LID range assigned to each GUID

3.2.2.1.2 Signaling

When OpenSM receives a HUP signal, it starts a new heavy sweep as if a trap has been received or a topology change has been found.

Also, SIGUSR1 can be used to trigger a reopen of `/var/log/opensm.log` for logrotate purposes.

3.2.2.1.3 Running opensm

The defaults of **opensm** were designed to meet the common case usage on clusters with up to a few hundred nodes. Thus, in this default mode, **opensm** will scan the IB fabric, initialize it, and sweep occasionally for changes.

To run **opensm** in the default mode, simply enter:

```
host1# opensm
```

Note that **opensm** needs to be run on at least one machine in an IB subnet.

By default, an **opensm** run is logged to two files: `/var/log/messages` and `/var/log/opensm.log`. The first file, `message`, registers only general major events; the second file, `opensm.log`, includes details of reported errors. All errors reported in `opensm.log` should be treated as indicators of IB fabric health. Both log files should include the message “SUBNET UP” if opensm was able to setup the subnet correctly.

If a fatal, non-recoverable error occurs, opensm exits.

3.2.2.1.4 Running OpenSM As Daemon

OpenSM can also run as daemon. To run OpenSM in this mode, enter:

```
host1# /etc/init.d/opensmd start
```

3.2.2.2 osmtest

osmtest is a test program for validating the InfiniBand Subnet Manager and Subnet Administrator. **osmtest** provides a test suite for **opensm**. It can create an inventory file of all available nodes, ports, and PathRecords, including all their fields. It can also verify the existing inventory with all the object fields, and matches it to a pre-saved one. See Section 3.2.2.2.1.

osmtest has the following test flows:

- Multicast Compliancy test
- Event Forwarding test
- Service Record registration test
- RMPP stress test
- Small SA Queries stress test

Syntax

```
osmtest [OPTIONS]
```

For the complete list of osmtest options, please run:

```
osmtest--help / -h / -?
```

3.2.2.2.1 Running osmtest

To run **osmtest** in the default mode, simply enter:

```
host1# osmtest
```

The default mode runs all the flows except for the Quality of Service flow (see Section 3.2.2.6).

After installing **opensm** (and if the InfiniBand fabric is stable), it is recommended to run the following command in order to generate the inventory file:

```
host1# osmtest -f c
```

Immediately afterwards, run the following command to test **opensm**:

```
host1# osmtest -f a
```

Finally, it is recommended to occasionally run "**`osmtest -v`**" (with verbosity) to verify that nothing in the fabric has changed.

3.2.2.3 Partitions

OpenSM enables the configuration of partitions (PKeys) in an InfiniBand fabric. By default, OpenSM searches for the partitions configuration file under the name `/usr/etc/opensm/partitions.conf`. To change this filename, you can use **opensm** with the '--Pconfig' or '-P' flags.

The default partition is created by OpenSM unconditionally, even when a partition configuration file does not exist or cannot be accessed.

The default partition has a P_Key value of 0x7fff. The port out of which runs OpenSM is assigned full membership in the default partition. All other end-ports are assigned partial membership.

3.2.2.3.1 File Format

Notes:

- Line content followed after '#' character is comment and ignored by parser.

General File Format

```
<Partition Definition>:[<newline>]<Partition Properties>
```

- <Partition Definition>:

```
[PartitionName][=PKey][,ipoib_bc_flags][,defmember=full|limited]
```

where

PartitionName	String, will be used with logging. When omitted empty string will be used.
PKey	P_Key value for this partition. Only low 15 bits will be used. When omitted will be auto-generated.
ipoib_bc_flags	Used to indicate/specify IPoIB capability of this partition.
defmember=full|limited|both	Specifies default membership for port GUID list. Default is limited.

ipoib_bc_flags are:

ipoib	Indicates that this partition may be used for IPoIB, as a result the IPoIB broadcast group will be created with the flags given, if any.
rate=<val>	Specifies rate for this IPoIB MC group (default is 3 (10GBps))
mtu=<val>	Specifies MTU for this IPoIB MC group (default is 4 (2048))
sl=<val>	Specifies SL for this IPoIB MC group (default is 0)
scope=<val>	Specifies scope for this IPoIB MC group (default is 2 (link local))

- <Partition Properties>:

```
[<Port list>|<MCast Group>]* | <Port list>
```

  - <Port List>:

```
<Port Specifier>[,<Port Specifier>]
```

    - <Port Specifier>:

```
<PortGUID>[=[full|limited|both]]
```

    where

PortGUID	GUID of partition member EndPort. Hexadecimal numbers should start from 0x, decimal numbers are accepted too.
full, limited	Indicates full and/or limited membership for this both port. When omitted (or unrecognized) limited membership is assumed. Both indicates both full and limited membership for this port.

  - <MCast Group>:

```
mgid=gid[,mgroup_flag]*<newline>
```

  where

mgid=gid	gid specified is verified to be a Multicast address IP groups are verified to match the rate and mtu of the broadcast group. The P_Key bits of the mgid for IP groups are verified to either match the P_Key specified in by "Partition Definition" or if they are 0x0000 the P_Key will be copied into those bits.	
mgroup_flag	rate=<val>	Specifies rate for this MC group (default is 3 (10GBps))
	mtu=<val>	Specifies MTU for this MC group (default is 4 (2048))
	sl=<val>	Specifies SL for this MC group (default is 0)
	scope=<val>	Specifies scope for this MC group (default is 2 (link local)). Multiple scope settings are permitted for a partition. NOTE: This overwrites the scope nibble of the specified mgid. Furthermore specifying multiple scope settings will result in multiple MC groups being created.
	qkey=<val>	Specifies the Q_Key for this MC group (default: 0x0b1b for IP groups, 0 for other groups)
	tclass=<val>	Specifies tclass for this MC group (default is 0)
	FlowLabel=<val>	Specifies FlowLabel for this MC group (default is 0)

Note that values for rate, MTU, and scope should be specified as defined in the IBTA specification (for example, mtu=4 for 2048). To use 4K MTU, edit that entry to `"mtu=5"` (5 indicates 4K MTU to that specific partition).

PortGUIDs list:

```
PortGUID         GUID of partition member EndPort. Hexadecimal numbers should start
from 0x, decimal numbers are accepted too.
full or limited   indicates full or limited membership for this port. When omitted (or
unrecognized) limited membership is assumed.
```

There are some useful keywords for PortGUID definition:

- 'ALL' means all end ports in this subnet
- 'ALL_CAS' means all Channel Adapter end ports in this subnet
- 'ALL_VCAS' means all virtual end ports in the subnet
- 'ALL_SWITCHES' means all Switch end ports in this subnet
- 'ALL_ROUTERS' means all Router end ports in this subnet
- 'SELF' means subnet manager's port.An empty list means that there are no ports in this partition

Notes:

- White space is permitted between delimiters ('=', ',',':',';').
- PartitionName does not need to be unique, PKey does need to be unique. If PKey is repeated then those partition configurations will be merged and first PartitionName will be used (see also next note).
- It is possible to split partition configuration in more than one definition, but then PKey should be explicitly specified (otherwise different PKey values will be generated for those definitions).

Examples

```
Default=0x7fff : ALL, SELF=full ;
Default=0x7fff : ALL, ALL_SWITCHES=full, SELF=full ;

NewPartition , ipoib : 0x123456=full, 0x3456789034=limi, 0x2134af2306 ;

YetAnotherOne = 0x300 : SELF=full ;
YetAnotherOne = 0x300 : ALL=limited ;

ShareIO = 0x80 , defmember=full : 0x123451, 0x123452;
# 0x123453, 0x123454 will be limited
ShareIO = 0x80 : 0x123453, 0x123454, 0x123455=full;
# 0x123456, 0x123457 will be limited
ShareIO = 0x80 : defmember=limited : 0x123456, 0x123457, 0x123458=full;
ShareIO = 0x80 , defmember=full : 0x123459, 0x12345a;
ShareIO = 0x80 , defmember=full : 0x12345b, 0x12345c=limited, 0x12345d;

# multicast groups added to default
Default=0x7fff,ipoib:
mgid=ff12:401b::0707,sl=1 # random IPv4 group
mgid=ff12:601b::16 # MLDv2-capable routers
mgid=ff12:401b::16 # IGMP
mgid=ff12:601b::2 # All routers
```

```
mgid=ff12::1,sl=1,Q_Key=0xDEADBEEF,rate=3,mtu=2 # random group
ALL=full;
```

The following rule is equivalent to how OpenSM used to run prior to the partition manager:

```
Default=0x7fff,ipoib:ALL=full;
```

3.2.2.4 Effect of Topology Changes

If a link is added or removed, OpenSM may not recalculate the routes that do not have to change. A route has to change if the port is no longer UP or no longer the MinHop. When routing changes are performed, the same algorithm for balancing the routes is invoked.

In the case of using the file based routing, any topology changes are currently ignored. The 'file' routing engine just loads the LFTs from the file specified, with no reaction to real topology. Obviously, this will not be able to recheck LIDs (by GUID) for disconnected nodes, and LFTs for non-existent switches will be skipped. Multicast is not affected by 'file' routing engine (this uses min hop tables).

3.2.2.5 Routing Algorithms

OpenSM offers the following routing engines:

1. "Min Hop Algorithm"

   Based on the minimum hops to each node where the path length is optimized.

2. "UPDN Algorithm"

   Based on the minimum hops to each node, but it is constrained to ranking rules. This algorithm should be chosen if the subnet is not a pure Fat Tree, and a deadlock may occur due to a loop in the subnet.

3. "Fat-tree Routing Algorithm"

   This algorithm optimizes routing for a congestion-free "shift" communication pattern. It should be chosen if a subnet is a symmetrical Fat Tree of various types, not just a K-ary-N-Tree: non-constant K, not fully staffed, and for any CBB ratio. Similar to UPDN, Fat Tree routing is constrained to ranking rules.

4. "LASH Routing Algorithm"

   Uses InfiniBand virtual layers (SL) to provide deadlock-free shortest-path routing while also distributing the paths between layers. LASH is an alternative deadlock-free, topology-agnostic routing algorithm to the non-minimal UPDN algorithm. It avoids the use of a potentially congested root node.

5. "DOR Routing Algorithm"

   Based on the Min Hop algorithm, but avoids port equalization except for redundant links between the same two switches. This provides deadlock free routes for hypercubes when the fabric is cabled as a hypercube and for meshes when cabled as a mesh.

6. "Torus-2QoS Routing Algorithm"

   Based on the DOR Unicast routing algorithm specialized for 2D/3D torus topologies. Torus-2QoS provides deadlock-free routing while supporting two quality of service (QoS) levels. Additionally, it can route around multiple failed fabric links or a single failed fabric switch without introducing deadlocks, and without changing path SL values granted before the failure.

7. "Unicast Routing Cache"

Unicast routing cache prevents routing recalculation (which is a heavy task in a large cluster) when no topology change was detected during the heavy sweep, or when the topology change does not require new routing calculation (for example, when one or more CAs/RTRs/leaf switches going down, or one or more of these nodes coming back after being down).

8. "Routing Chains"

   Allows routing configuration of different parts of a single InfiniBand subnet by different routing engines. In the current release, minhop/updn/ftree/dor/torus-2QoS/pqft can be combined.

OpenSM also supports a file method which can load routes from a table – see Modular Routing Engine below.

MINHOP/UPDN/DOR routing algorithms are comprised of two stages:

1. MinHop matrix calculation. How many hops are required to get from each port to each LID. The algorithm to fill these tables is different if you run standard (min hop) or Up/Down. For standard routing, a "relaxation" algorithm is used to propagate min hop from every destination LID through neighbor switches. For Up/Down routing, a BFS from every target is used. The BFS tracks link direction (up or down) and avoid steps that will perform up after a down step was used.
2. Once MinHop matrices exist, each switch is visited and for each target LID a decision is made as to what port should be used to get to that LID. This step is common to standard and Up/Down routing. Each port has a counter counting the number of target LIDs going through it. When there are multiple alternative ports with same MinHop to a LID, the one with less previously assigned ports is selected.

   If LMC > 0, more checks are added. Within each group of LIDs assigned to same target port:

   a. Use only ports which have same MinHop
   b. First prefer the ones that go to different systemImageGuid (then the previous LID of the same LMC group)
   c. If none, prefer those which go through another NodeGuid
   d. Fall back to the number of paths method (if all go to same node).

3.2.2.5.1 Min Hop Algorithm

The Min Hop algorithm is invoked by default if no routing algorithm is specified. It can also be invoked by specifying '-R minhop'.

The Min Hop algorithm is divided into two stages: computation of min-hop tables on every switch and LFT output port assignment. Link subscription is also equalized with the ability to override based on port GUID. The latter is supplied by:

```
-i <equalize-ignore-guids-file>
-ignore-guids <equalize-ignore-guids-file>
                 This option provides the means to define a set of ports (by guids)
that will be ignored by the link load equalization algorithm.
```

LMC awareness routes based on (remote) system or switch basis.

3.2.2.5.2 UPDN Algorithm

The UPDN algorithm is designed to prevent deadlocks from occurring in loops of the subnet. A loop-deadlock is a situation in which it is no longer possible to send data between any two hosts connected through the loop. As such, the UPDN routing algorithm should be send if the subnet is not a pure Fat Tree, and one of its loops may experience a deadlock (due, for example, to high pressure).

The UPDN algorithm is based on the following main stages:

1. Auto-detect root nodes - based on the CA hop length from any switch in the subnet, a statistical histogram is built for each switch (hop num vs number of occurrences). If the histogram reflects a specific column (higher than others) for a certain node, then it is marked as a root node. Since the algorithm is statistical, it may not find any root nodes. The list of the root nodes found by this auto-detect stage is used by the ranking process stage.

> The user can override the node list manually

> If this stage cannot find any root nodes, and the user did not specify a guid list file, OpenSM defaults back to the Min Hop routing algorithm.

2. Ranking process - All root switch nodes (found in stage 1) are assigned a rank of 0. Using the BFS algorithm, the rest of the switch nodes in the subnet are ranked incrementally. This ranking aids in the process of enforcing rules that ensure loop-free paths.
3. Min Hop Table setting - after ranking is done, a BFS algorithm is run from each (CA or switch) node in the subnet. During the BFS process, the FDB table of each switch node traversed by BFS is updated, in reference to the starting node, based on the ranking rules and guid values.

At the end of the process, the updated FDB tables ensure loop-free paths through the subnet.

> Up/Down routing does not allow LID routing communication between switches that are located inside spine “switch systems”. The reason is that there is no way to allow a LID route between them that does not break the Up/Down rule. One ramification of this is that you cannot run SM on switches other than the leaf switches of the fabric.

UPDN Algorithm Usage

Activation through OpenSM

- Use '-R updn' option (instead of old '-u') to activate the UPDN algorithm.
- Use '-a <root_guid_file>' for adding an UPDN guid file that contains the root nodes for ranking. If the `-a' option is not used, OpenSM uses its auto-detect root nodes algorithm.

Notes on the guid list file:

- A valid guid file specifies one guid in each line. Lines with an invalid format will be discarded
- The user should specify the root switch guids

3.2.2.5.3 Fat-tree Routing Algorithm

The fat-tree algorithm optimizes routing for "shift" communication pattern. It should be chosen if a subnet is a symmetrical or almost symmetrical fat-tree of various types. It supports not just K-ary-N-Trees, by handling for non-constant K, cases where not all leafs (CAs) are present, any

Constant Bisectional Ratio (CBB)ratio. As in UPDN, fat-tree also prevents credit-loop-deadlocks.

If the root guid file is not provided ('**-a**' or '**--root_guid_file**' options), the topology has to be pure fat-tree that complies with the following rules:

- Tree rank should be between two and eight (inclusively)
- Switches of the same rank should have the same number of UP-going port groups[1], unless they are root switches, in which case the shouldn't have UP-going ports at all.
- Switches of the same rank should have the same number of DOWN-going port groups, unless they are leaf switches.
- Switches of the same rank should have the same number of ports in each UP-going port group.
- Switches of the same rank should have the same number of ports in each DOWN-going port group.
- All the CAs have to be at the same tree level (rank).

If the root guid file is provided, the topology does not have to be pure fat-tree, and it should only comply with the following rules:

- Tree rank should be between two and eight (inclusively)
- All the Compute Nodes[2] have to be at the same tree level (rank). Note that non-compute node CAs are allowed here to be at different tree ranks.

Topologies that do not comply cause a fallback to min hop routing. Note that this can also occur on link failures which cause the topology to no longer be a “pure” fat-tree.

Note that although fat-tree algorithm supports trees with non-integer CBB ratio, the routing will not be as balanced as in case of integer CBB ratio. In addition to this, although the algorithm allows leaf switches to have any number of CAs, the closer the tree is to be fully populated, the more effective the "shift" communication pattern will be. In general, even if the root list is provided, the closer the topology to a pure and symmetrical fat-tree, the more optimal the routing will be.

The algorithm also dumps compute node ordering file (`opensm-ftree-ca-order.dump`) in the same directory where the OpenSM log resides. This ordering file provides the CN order that may be used to create efficient communication pattern, that will match the routing tables.

Routing between non-CN Nodes

The use of the io_guid_file option allows non-CN nodes to be located on different levels in the fat tree. In such case, it is not guaranteed that the Fat Tree algorithm will route between two non-CN nodes. In the scheme below, N1, N2 and N3 are non-CN nodes. Although all the CN have routes to and from them, there will not necessarily be a route between N1,N2 and N3. Such routes would require to use at least one of the switches the wrong way around.

```
 Spine1   Spine2    Spine 3
  / \     /  |  \     /   \
 /   \   /   |   \   /     \
N1  Switch   N2  Switch     N3
     /|\          /|\
    / | \        / | \
```

1. Ports that are connected to the same remote switch are referenced as ‘port group’
2. List of compute nodes (CNs) can be specified by ‘-u’ or ‘--cn_guid_file’ OpenSM options.

```
    Going down to compute nodes
```

To solve this problem, a list of non-CN nodes can be specified by \'-G\' or \'--io_guid_file\' option. These nodes will be allowed to use switches the wrong way around a specific number of times (specified by \'-H\' or \'--max_reverse_hops\'. With the proper max_reverse_hops and io_guid_file values, you can ensure full connectivity in the Fat Tree. In the scheme above, with a max_reverse_hop of 1, routes will be instanciated between N1<->N2 and N2<->N3. With a max_reverse_hops value of 2, N1,N2 and N3 will all have routes between them.

Using max_reverse_hops creates routes that use the switch in a counter-stream way. This option should never be used to connect nodes with high bandwidth traffic between them! It should only be used to allow connectivity for HA purposes or similar. Also having routes the other way around can cause credit loops.

Activation through OpenSM

- Use ‘-R ftree’ option to activate the fat-tree algorithm

LMC > 0 is not supported by fat-tree routing. If this is specified, the default routing algorithm is invoked instead.

3.2.2.5.4 LASH Routing Algorithm

LASH is an acronym for LAyered SHortest Path Routing. It is a deterministic shortest path routing algorithm that enables topology agnostic deadlock-free routing within communication networks.

When computing the routing function, LASH analyzes the network topology for the shortest-path routes between all pairs of sources / destinations and groups these paths into virtual layers in such a way as to avoid deadlock.

LASH analyzes routes and ensures deadlock freedom between switch pairs. The link from HCA between and switch does not need virtual layers as deadlock will not arise between switch and HCA.

In more detail, the algorithm works as follows:

1. LASH determines the shortest-path between all pairs of source / destination switches. Note, LASH ensures the same SL is used for all SRC/DST - DST/SRC pairs and there is no guarantee that the return path for a given DST/SRC will be the reverse of the route SRC/DST.
2. LASH then begins an SL assignment process where a route is assigned to a layer (SL) if the addition of that route does not cause deadlock within that layer. This is achieved by maintaining and analysing a channel dependency graph for each layer. Once the potential addition of a path could lead to deadlock, LASH opens a new layer and continues the process.
3. Once this stage has been completed, it is highly likely that the first layers processed will contain more paths than the latter ones. To better balance the use of layers, LASH moves paths from one layer to another so that the number of paths in each layer averages out.

Note that the implementation of LASH in opensm attempts to use as few layers as possible. This number can be less than the number of actual layers available.

In general LASH is a very flexible algorithm. It can, for example, reduce to Dimension Order Routing in certain topologies, it is topology agnostic and fares well in the face of faults.

It has been shown that for both regular and irregular topologies, LASH outperforms Up/Down. The reason for this is that LASH distributes the traffic more evenly through a network, avoiding the bottleneck issues related to a root node and always routes shortest-path.

The algorithm was developed by Simula Research Laboratory.

Use '-R lash -Q' option to activate the LASH algorithm

QoS support has to be turned on in order that SL/VL mappings are used.

LMC > 0 is not supported by the LASH routing. If this is specified, the default routing algorithm is invoked instead.

For open regular cartesian meshes the DOR algorithm is the ideal routing algorithm. For toroidal meshes on the other hand there are routing loops that can cause deadlocks. LASH can be used to route these cases. The performance of LASH can be improved by preconditioning the mesh in cases where there are multiple links connecting switches and also in cases where the switches are not cabled consistently. To invoke this, use '-R lash -Q --do_mesh_analysis'. This will add an additional phase that analyses the mesh to try to determine the dimension and size of a mesh. If it determines that the mesh looks like an open or closed cartesian mesh it reorders the ports in dimension order before the rest of the LASH algorithm runs.

3.2.2.5.5 DOR Routing Algorithm

The Dimension Order Routing algorithm is based on the Min Hop algorithm and so uses shortest paths. Instead of spreading traffic out across different paths with the same shortest distance, it chooses among the available shortest paths based on an ordering of dimensions. Each port must be consistently cabled to represent a hypercube dimension or a mesh dimension. Paths are grown from a destination back to a source using the lowest dimension (port) of available paths at each step. This provides the ordering necessary to avoid deadlock. When there are multiple links between any two switches, they still represent only one dimension and traffic is balanced across them unless port equalization is turned off. In the case of hypercubes, the same port must be used throughout the fabric to represent the hypercube dimension and match on both ends of the cable. In the case of meshes, the dimension should consistently use the same pair of ports, one port on one end of the cable, and the other port on the other end, continuing along the mesh dimension.

Use '-R dor' option to activate the DOR algorithm.

3.2.2.5.6 Torus-2QoS Routing Algorithm

Torus-2QoS is a routing algorithm designed for large-scale 2D/3D torus fabrics. The torus-2QoS routing engine can provide the following functionality on a 2D/3D torus:

- Free of credit loops routing
- Two levels of QoS, assuming switches support 8 data VLs
- Ability to route around a single failed switch, and/or multiple failed links, without:
  - introducing credit loops

- changing path SL values
- Very short run times, with good scaling properties as fabric size increases

3.2.2.5.6.1 Unicast Routing Cache

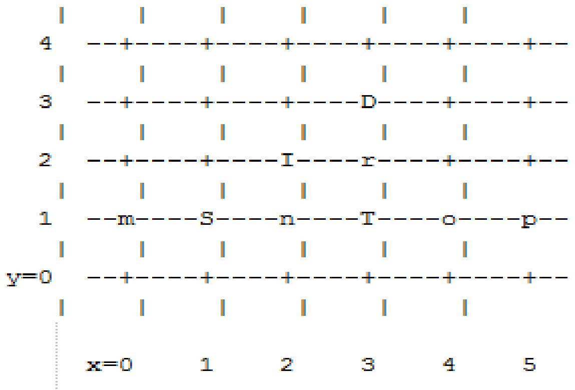
Torus-2 QoS is a DOR-based algorithm that avoids deadlocks that would otherwise occur in a torus using the concept of a dateline for each torus dimension. It encodes into a path SL which datelines the path crosses as follows:

```
sl = 0;
for (d = 0; d < torus_dimensions; d++)
/* path_crosses_dateline(d) returns 0 or 1 */
sl |= path_crosses_dateline(d) << d;
```

For a 3D torus, that leaves one SL bit free, which torus-2 QoS uses to implement two QoS levels. Torus-2 QoS also makes use of the output port dependence of switch SL2VL maps to encode into one VL bit the information encoded in three SL bits. It computes in which torus coordinate direction each inter-switch link "points", and writes SL2VL maps for such ports as follows:

```
for (sl = 0; sl < 16; sl ++)
/* cdir(port) reports which torus coordinate direction a switch port
* "points" in, and returns 0, 1, or 2 */
sl2vl(iport,oport,sl) = 0x1 & (sl >> cdir(oport));
```

Thus, on a pristine 3D torus, i.e., in the absence of failed fabric switches, torus-2 QoS consumes 8 SL values (SL bits 0-2) and 2 VL values (VL bit 0) per QoS level to provide deadlock-free routing on a 3D torus. Torus-2 QoS routes around link failure by "taking the long way around" any 1D ring interrupted by a link failure. For example, consider the 2D 6x5 torus below, where switches are denoted by [+a-zA-Z]:



```
     |    |    |    |    |    |
  4  --+----+----+----+----+----+--
     |    |    |    |    |    |
  3  --+----+----+----D----+----+--
     |    |    |    |    |    |
  2  --+----+----I----r----+----+--
     |    |    |    |    |    |
  1  --m----S----n----T----o----p--
     |    |    |    |    |    |
y=0  --+----+----+----+----+----+--
     |    |    |    |    |    |

     x=0  1    2    3    4    5
```

For a pristine fabric the path from S to D would be S-n-T-r-D. In the event that either link S-n or n-T has failed, torus-2QoS would use the path S-m-p-o-T-r-D.

Note that it can do this without changing the path SL value; once the 1D ring m-S-n-T-o-p-m has been broken by failure, path segments using it cannot contribute to deadlock, and the x-direction dateline (between, say, x=5 and x=0) can be ignored for path segments on that ring. One result of this is that torus-2QoS can route around many simultaneous link failures, as long as no 1D ring is broken into disjoint segments. For example, if links n-T and T-o have both failed, that ring has been broken into two disjoint segments, T and o-p-m-S-n. Torus-2QoS checks for such issues, reports if they are found, and refuses to route such fabrics.

Note that in the case where there are multiple parallel links between a pair of switches, torus-2QoS will allocate routes across such links in a round-robin fashion, based on ports at the path destination switch that are active and not used for inter-switch links. Should a link that is one of severalsuch parallel links fail, routes are redistributed across the remaining links. When the last of such a set of parallel links fails, traffic is rerouted as described above.

Handling a failed switch under DOR requires introducing into a path at least one turn that would be otherwise "illegal", i.e. not allowed by DOR rules. Torus-2QoS will introduce such a turn as close as possible to the failed switch in order to route around it. n the above example, suppose switch T has failed, and consider the path from S to D. Torus-2QoS will produce the path S-n-I-r-D, rather than the S-n-T-r-D path for a pristine torus, by introducing an early turn at n. Normal DOR rules will cause traffic arriving at switch I to be forwarded to switch r; for traffic arriving from I due to the "early" turn at n, this will generate an "illegal" turn at I.

Torus-2QoS will also use the input port dependence of SL2VL maps to set VL bit 1 (which would be otherwise unused) for y-x, z-x, and z-y turns, i.e., those turns that are illegal under DOR. This causes the first hop after any such turn to use a separate set of VL values, and prevents deadlock in the presence of a single failed switch. For any given path, only the hops after a turn that is illegal under DOR can contribute to a credit loop that leads to deadlock. So in the example above with failed switch T, the location of the illegal turn at I in the path from S to D requires that any credit loop caused by that turn must encircle the failed switch at T. Thus the second and later hops after the illegal turn at I (i.e., hop r-D) cannot contribute to a credit loop because they cannot be used to construct a loop encircling T. The hop I-r uses a separate VL, so it cannot contribute to a credit loop encircling T. Extending this argument shows that in addition to being capable of routing around a single switch failure without introducing deadlock, torus-2QoS can also route around multiple failed switches on the condition they are adjacent in the last dimension routed by DOR. For example, consider the following case on a 6x6 2D torus:

```
      |    |    |    |    |    |
  5 --+----+----+----+----+----+--
      |    |    |    |    |    |
  4 --+----+----+----D----+----+--
      |    |    |    |    |    |
  3 --+----+----I----u----+----+--
      |    |    |    |    |    |
  2 --+----+----q----R----+----+--
      |    |    |    |    |    |
  1 --m----S----n----T----o----p--
      |    |    |    |    |    |
y=0 --+----+----+----+----+----+--
      |    |    |    |    |    |

    x=0    1    2    3    4    5
```

Suppose switches T and R have failed, and consider the path from S to D. Torus-2QoS will generate the path S-n-q-I-u-D, with an illegal turn at switch I, and with hop I-u using a VL with bit 1 set. As a further example, consider a case that torus-2QoS cannot route without deadlock: two failed switches adjacent in a dimension that is not the last dimension routed by DOR; here the failed switches are O and T:

```
      |    |    |    |    |    |
 5  --+----+----+----+----+----+--
      |    |    |    |    |    |
 4  --+----+----+----+----+----+--
      |    |    |    |    |    |
 3  --+----+----+----+----D----+--
      |    |    |    |    |    |
 2  --+----+----I----q----r----+--
      |    |    |    |    |    |
 1  --m----S----n----O----T----p--
      |    |    |    |    |    |
y=0 --+----+----+----+----+----+--
      |    |    |    |    |    |

    x=0    1    2    3    4    5
```

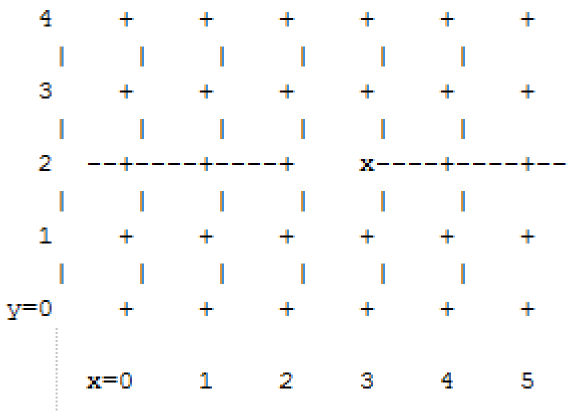
In a pristine fabric, torus-2QoS would generate the path from S to D as S-n-O-T-r-D. With failed switches O and T, torus-2QoS will generate the path S-n-I-q-r-D, with illegal turn at switch I, and with hop I-q using a VL with bit 1 set. In contrast to the earlier examples, the second hop after the illegal turn, q-r, can be used to construct a credit loop encircling the failed switches.

3.2.2.5.6.2 Multicast Routing

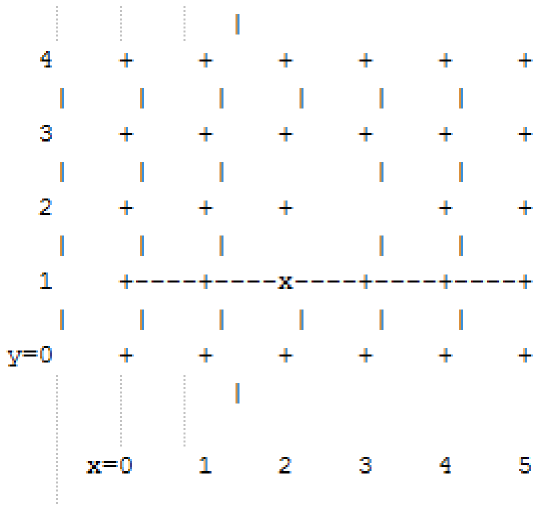
Since torus-2QoS uses all four available SL bits, and the three data VL bits that are typically available in current switches, there is no way to use SL/VL values to separate multicast traffic from unicast traffic. Thus, torus-2QoS must generate multicast routing such that credit loops cannot arise from a combination of multicast and unicast path segments. It turns out that it is possible to construct spanning trees for multicast routing that have that property. For the 2D 6x5 torus example above, here is the full-fabric spanning tree that torus-2QoS will construct, where "x" is the root switch and each "+" is a non-root switch:

```
 4    +    +    +    +    +    +
      |    |    |    |    |    |
 3    +    +    +    +    +    +
      |    |    |    |    |    |
 2    +----+----+----x----+----+
      |    |    |    |    |    |
 1    +    +    +    +    +    +
      |    |    |    |    |    |
y=0   +    +    +    +    +    +

    x=0    1    2    3    4    5
```

For multicast traffic routed from root to tip, every turn in the above spanning tree is a legal DOR turn. For traffic routed from tip to root, and some traffic routed through the root, turns are not legal DOR turns. However, to construct a credit loop, the union of multicast routing on this spanning tree with DOR unicast routing can only provide 3 of the 4 turns needed for the loop. In addition, if none of the above spanning tree branches crosses a dateline used for unicast credit loop avoidance on a torus, and if multicast traffic is confined to SL 0 or SL 8 (recall that torus-2QoS uses SL bit 3 to differentiate QoS level), then multicast traffic also cannot contribute to the "ring" credit loops that are otherwise possible in a torus. Torus-2QoS uses these ideas to create a master spanning tree. Every multicast group spanning tree will be constructed as a subset of the master tree, with the same root as the master tree. Such multicast group spanning trees will in general not be optimal for groups which are a subset of the full fabric. However, this compromise must be made to enable support for two QoS levels on a torus while preventing credit loops. In the presence of link or switch failures that result in a fabric for which torus-2QoS can generate credit-loop-free unicast routes, it is also possible to generate a master spanning tree for multicast that retains the required properties. For example, consider that same 2D 6x5 torus, with the link from (2,2) to (3,2) failed. Torus-2QoS will generate the following master spanning tree:



Two things are notable about this master spanning tree. First, assuming the x dateline was between x=5 and x=0, this spanning tree has a branch that crosses the dateline. However, just as for unicast, crossing a dateline on a 1D ring (here, the ring for y=2) that is broken by a failure cannot contribute to a torus credit loop. Second, this spanning tree is no longer optimal even for multicast groups that encompass the entire fabric. That, unfortunately, is a compromise that must be made to retain the other desirable properties of torus-2QoS routing. In the event that a single switch fails, torus-2QoS will generate a master spanning tree that has no "extra" turns by appropriately selecting a root switch. In the 2D 6x5 torus example, assume now that the switch at (3,2), i.e. the root for a pristine fabric, fails. Torus-2QoS will generate the following master spanning tree for that case:



Assuming the y dateline was between y=4 and y=0, this spanning tree has a branch that crosses a dateline. However, again this cannot contribute to credit loops as it occurs on a 1D ring (the ring for x=3) that is broken by a failure, as in the above example.

3.2.2.5.6.3 Torus Topology Discovery

The algorithm used by torus-2QoS to construct the torus topology from the undirected graph representing the fabric requires that the radix of each dimension be configured via torus-2QoS.conf. It also requires that the torus topology be "seeded"; for a 3D torus this requires configuring four switches that define the three coordinate directions of the torus. Given this starting information, the algorithm is to examine the cube formed by the eight switch locations bounded by the corners (x,y,z) and (x+1,y+1,z+1). Based on switches already placed into the torus topology at some of these locations, the algorithm examines 4-loops of interswitch links to find the one that is consistent with a face of the cube of switch locations, and adds its switches to the discovered topology in the correct locations.

Because the algorithm is based on examining the topology of 4-loops of links, a torus with one or more radix-4 dimensions requires extra initial seed configuration. See torus-2QoS.conf(5) for details. Torus-2QoS will detect and report when it has insufficient configuration for a torus with radix-4 dimensions.

In the event the torus is significantly degraded, i.e., there are many missing switches or links, it may happen that torus-2QoS is unable to place into the torus some switches and/or links that were discovered in the fabric, and will generate a warning in that case. A similar condition occurs if torus-2QoS is mis-configured, i.e., the radix of a torus dimension as configured does not match the radix of that torus dimension as wired, and many switches/links in the fabric will not be placed into the torus.

3.2.2.5.6.4 Quality Of Service Configuration

OpenSM will not program switches and channel adapters with SL2VL maps or VL arbitration configuration unless it is invoked with -Q. Since torus-2QoS depends on such functionality for correct operation, always invoke OpenSM with -Q when torus-2QoS is in the list of routing engines. Any quality of service configuration method supported by OpenSM will work with torus-2QoS, subject to the following limitations and considerations. For all routing engines supported by OpenSM except torus-2QoS, there is a one-to-one correspondence between QoS level and SL. Torus-2QoS can only support two quality of service levels, so only the high-order bit of any SL value used for unicast QoS configuration will be honored by torus-2QoS. For multicast QoS configuration, only SL values 0 and 8 should be used with torus-2QoS.

Since SL to VL map configuration must be under the complete control of torus-2QoS, any configuration via qos_sl2vl, qos_swe_sl2vl, etc., must and will be ignored, and a warning will be generated. Torus-2QoS uses VL values 0-3 to implement one of its supported QoS levels, and VL values 4-7 to implement the other. Hard-to-diagnose application issues may arise if traffic is not delivered fairly across each of these two VL ranges. Torus-2QoS will detect and warn if VL arbitration is configured unfairly across VLs in the range 0-3, and also in the range 4-7. Note that the default OpenSM VL arbitration configuration does not meet this constraint, so all torus-2QoS users should configure VL arbitration via qos_vlarb_high, qos_vlarb_low, etc.

Operational Considerations

Any routing algorithm for a torus IB fabric must employ path SL values to avoid credit loops. As a result, all applications run over such fabrics must perform a path record query to obtain the correct path SL for connection setup. Applications that use rdma_cm for connection setup will automatically meet this requirement.

If a change in fabric topology causes changes in path SL values required to route without credit loops, in general all applications would need to repath to avoid message deadlock. Since torus-2QoS has the ability to reroute after a single switch failure without changing path SL values, repathing by running applications is not required when the fabric is routed with torus-2QoS.

Torus-2QoS can provide unchanging path SL values in the presence of subnet manager failover provided that all OpenSM instances have the same idea of dateline location. See torus-2QoS.conf(5) for details. Torus-2QoS will detect configurations of failed switches and links that prevent routing that is free of credit loops, and will log warnings and refuse to route. If "no_fallback" was configured in the list of OpenSM routing engines, then no other routing engine will attempt to route the fabric. In that case all paths that do not transit the failed components will continue to work, and the subset of paths that are still operational will continue to remain free of credit loops. OpenSM will continue to attempt to route the fabric after every sweep interval, and after any change (such as a link up) in the fabric topology. When the fabric components are repaired, full functionality will be restored. In the event OpenSM was configured to allow some other engine to route the fabric if torus-2QoS fails, then credit loops and message deadlock are likely if torus-2QoS had previously routed the fabric successfully. Even if the other engine is capable of routing a torus without credit loops, applications that built connections with path SL values granted under torus-2QoS will likely experience message deadlock under routing generated by a different engine, unless they repath. To verify that a torus fabric is routed free of credit loops, use ibdmchk to analyze data collected via ibdiagnet -vlr.

3.2.2.5.6.5 Torus-2QoS Configuration File Syntax

The file torus-2QoS.conf contains configuration information that is specific to the OpenSM routing engine torus-2QoS. Blank lines and lines where the first non-whitespace character is "#" are ignored. A token is any contiguous group of non-whitespace characters. Any tokens on a line following the recognized configuration tokens described below are ignored.

```
[torus|mesh] x_radix[m|M|t|T] y_radix[m|M|t|T] z_radix[m|M|t|T]
```

Either torus or mesh must be the first keyword in the configuration, and sets the topology that torus-2QoS will try to construct. A 2D topology can be configured by specifying one of x_radix, y_radix, or z_radix as 1. An individual dimension can be configured as mesh (open) or torus (looped) by suffixing its radix specification with one of m, M, t, or T. Thus, "mesh 3T 4 5" and "torus 3 4M 5M" both specify the same topology.

Note that although torus-2QoS can route mesh fabrics, its ability to route around failed components is severely compromised on such fabrics. A failed fabric components very likely to cause a disjoint ring; see UNICAST ROUTING in torus-2QoS(8).

```
xp_link sw0_GUID sw1_GUID
yp_link sw0_GUID sw1_GUID
zp_link sw0_GUID sw1_GUID
xm_link sw0_GUID sw1_GUID
ym_link sw0_GUID sw1_GUID
zm_link sw0_GUID sw1_GUID
```

These keywords are used to seed the torus/mesh topology. For example, "xp_link 0x2000 0x2001" specifies that a link from the switch with node GUID 0x2000 to the switch with node GUID 0x2001 would point in the positive x direction, while "xm_link 0x2000 0x2001" specifies that a link from the switch with node GUID 0x2000 to the switch with node GUID 0x2001 would point in the negative x direction. All the link keywords for a given seed must specify the same "from" switch.

In general, it is not necessary to configure both the positive and negative directions for a given coordinate; either is sufficient. However, the algorithm used for topology discovery needs extra information for torus dimensions of radix four (see TOPOLOGY DISCOVERY in torus-2QoS(8)). For such cases both the positive and negative coordinate directions must be specified.

Based on the topology specified via the torus/mesh keyword, torus-2QoS will detect and log when it has insufficient seed configuration.

```
x_dateline position
y_dateline position
z_dateline position
```

In order for torus-2QoS to provide the guarantee that path SL values do not change under any conditions for which it can still route the fabric, its idea of dateline position must not change relative to physical switch locations. The dateline keywords provide the means to configure such behavior.

The dateline for a torus dimension is always between the switch with coordinate 0 and the switch with coordinate radix-1 for that dimension. By default, the common switch in a torus seed is taken as the origin of the coordinate system used to describe switch location. The position parameter for a dateline keyword moves the origin (and hence the dateline) the specified amount relative to the common switch in a torus seed.

```
next_seed
```

If any of the switches used to specify a seed were to fail torus-2QoS would be unable to complete topology discovery successfully. The next_seed keyword specifies that the following link and dateline keywords apply to a new seed specification.

For maximum resiliency, no seed specification should share a switch with any other seed specification. Multiple seed specifications should use dateline configuration to ensure that torus-2QoS can grant path SL values that are constant, regardless of which seed was used to initiate topology discovery.

portgroup_max_ports max_ports - This keyword specifies the maximum number of parallel inter-switch links, and also the maximum number of host ports per switch, that torus-2QoS can accommodate. The default value is 16. Torus-2QoS will log an error message during topology discovery if this parameter needs to be increased. If this keyword appears multiple times, the last instance prevails.

port_order p1 p2 p3 ... - This keyword specifies the order in which CA ports on a destination switch are visited when computing routes. When the fabric contains switches connected with multiple parallel links, routes are distributed in a round-robin fashion across such links, and so changing the order that CA ports are visited changes the distribution of routes across such links. This may be advantageous for some specific traffic patterns.

The default is to visit CA ports in increasing port order on destination switches. Duplicate values in the list will be ignored.

EXAMPLE

```
# Look for a 2D (since x radix is one) 4x5 torus.
torus 1 4 5
# y is radix-4 torus dimension, need both
# ym_link and yp_link configuration.
yp_link 0x200000 0x200005 # sw @ y=0,z=0 -> sw @ y=1,z=0
ym_link 0x200000 0x20000f # sw @ y=0,z=0 -> sw @ y=3,z=0
# z is not radix-4 torus dimension, only need one of
# zm_link or zp_link configuration.
zp_link 0x200000 0x200001 # sw @ y=0,z=0 -> sw @ y=0,z=1
next_seed
yp_link 0x20000b 0x200010 # sw @ y=2,z=1 -> sw @ y=3,z=1
ym_link 0x20000b 0x200006 # sw @ y=2,z=1 -> sw @ y=1,z=1
zp_link 0x20000b 0x20000c # sw @ y=2,z=1 -> sw @ y=2,z=2
y_dateline -2 # Move the dateline for this seed
z_dateline -1 # back to its original position.
# If OpenSM failover is configured, for maximum resiliency
# one instance should run on a host attached to a switch
# from the first seed, and another instance should run
# on a host attached to a switch from the second seed.
# Both instances should use this torus-2QoS.conf to ensure
# path SL values do not change in the event of SM failover.
# port_order defines the order on which the ports would be
# chosen for routing.
port_order 7 10 8 11 9 12 25 28 26 29 27 30
```

3.2.2.5.7 Routing Chains

The routing chains feature is offering a solution that enables one to configure different parts of the fabric and define a different routing engine to route each of them. The routings are done in a sequence (hence the name "chains") and any node in the fabric that is configured in more than one part is left with the routing updated by the last routing engine it was a part of.

Configuring Routing Chains

The configuration for the routing chains feature consists of the following steps:

1. Define the port groups.
2. Define topologies based on previously defined port groups.
3. Define configuration files for each routing engine.
4. Define routing engine chains over previously defined topologies and configuration files.

Defining Port Groups

The basic idea behind the port groups is the ability to divide the fabric into sub-groups and give each group an identifier that can be used to relate to all nodes in this group. The port groups is a separate feature from the routing chains, but is a mandatory prerequisite for it. In addition, it is used to define the participants in each of the routing algorithms.

Defining a Port Group Policy File

In order to define a port group policy file, set the parameter 'pgrp_policy_file' in the opensm configuration file.

```
pgrp_policy_file /etc/opensm/conf/port_groups_policy_file
```

Configuring a Port Group Policy

The port groups policy file details the port groups in the fabric. The policy file should be composed of one or more paragraphs that define a group. Each paragraph should begin with the line 'port-group' and end with the line 'end-port-group'.

For example:

```
port-group
…port group qualifiers…
end-port-group
```

Port Group Qualifiers

Unlike the port group's beginning and ending which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. An inclusion of a colon in the name or the use of a port group, will result in the policy's failure.

Rule Qualifier

Parameter	Description	Example
name	Each group must have a name. Without a name qualifier, the policy fails.	name: grp1
use	'use' is an optional qualifier that one can define in order to describe the usage of this port group (if undefined, an empty string is used as a default).	use: first port group

There are several qualifiers used to describe a rule that determines which ports will be added to the group. Each port group may include one or more rules out of the rules described in the below table (At least one rule must be defined for each port group).

Parameter	Description	Example
guid list	Comma separated list of guids to include in the group. If no specific physical ports were configured, all physical ports of the guid are chosen. However, for each guid, one can detail specific physical ports to be included in the group. This can be done using the following syntax: • Specify a specific port in a guid to be chosen port-guid: 0x283@3 • Specify a specific list of ports in a guid to be chosen port-guid: 0x286@1/5/7 • Specify a specific range of ports in a guid to be chosen port-guid: 0x289@2-5 • Specify a list of specific ports and ports ranges in a guid to be chosen port-guid: 0x289@2-5/7/9-13/18 • Complex rule port-guid: 0x283@5-8/12/14, 0x286, 0x289/6/8/12	port-guid: 0x283, 0x286, 0x289
port guid range	It is possible to configure a range of guids to be chosen to the group. However, while using the range qualifier, it is impossible to detail specific physical ports. **Note:** A list of ranges cannot be specified. The below example is invalid and will cause the policy to fail: port-guid-range: 0x283-0x289, 0x290-0x295	port-guid-range: 0x283-0x289

Parameter	Description	Example
port name	One can configure a list of hostnames as a rule. Hosts with a node description that is built out of these hostnames will be chosen. Since the node description contains the network card index as well, one might also specify a network card index and a physical port to be chosen. For example, the given configuration will cause only physical port 2 of a host with the node description 'kuku HCA-1' to be chosen. `port` and `hca_idx` parameters are optional. If the port is unspecified, all physical ports are chosen. If `hca_idx` is unspecified, all card numbers are chosen. Specifying a hostname is mandatory. One can configure a list of `hostname/port/hca_idx` sets in the same qualifier as follows: port-name: hostname=kuku; port=2; hca_idx=1 , hostname=host1; port=3, hostname=host2 **Note:** `port-name` qualifier is not relevant for switches, but for HCA's only.	port-name: hostname=kuku; port=2; hca_idx=1
port regexp	One can define a regular expression so that only nodes with a matching node description will be chosen to the group[1]	port-regexp: SW
	It is possible to specify one physical port to be chosen for matching nodes (there is no option to define a list or a range of ports). The given example will cause only nodes that match physical port 3 to be added to the group.	port-regexp: SW:3
union rule	It is possible to define a rule that unites two different port groups. This means that all ports from both groups will be included in the united group.	union-rule: grp1, grp2

Parameter	Description	Example
subtract rule	One can define a rule that subtracts one port group from another. The given rule, for example, will cause all the ports which are a part of grp1, but not included in grp2, to be chosen. In subtraction (unlike union), the order does matter, since the purpose is to subtract the second group from the first one. There is no option to define more than two groups for union/subtraction. However, one can unite/subtract groups which are a union or a subtraction themselves, as shown in the port groups policy file example.	subtract-rule: grp1, grp2

1. This example shows how to choose nodes which their node description starts with 'SW'.

Predefined Port Groups

There are 3 predefined, automatically created port groups that are available for use, yet cannot be defined in the policy file (if a group in the policy is configured with the name of one of these predefined groups, the policy fails) -

- ALL - a group that includes all nodes in the fabric
- ALL_SWITCHES - a group that includes all switches in the fabric.
- ALL_CAS - a group that includes all HCA's in the fabric.

Port Groups Policy Examples

```
port-group
name: grp3
use: Subtract of groups grp1 and grp2
subtract-rule: grp1, grp2
end-port-group

port-group
name: grp1
port-guid: 0x281, 0x282, 0x283
end-port-group

port-group
name: grp2
port-guid-range: 0x282-0x286
port-name: hostname=server1 port=1
end-port-group

port-group
name: grp4
port-name: hostname=kika port=1 hca_idx=1
end-port-group

port-group
name: grp3
union-rule: grp3, grp4
end-port-group
```

Defining a Topologies Policy File

In order to define a port group policy file, set the parameter 'topo_policy_file' in the opensm configuration file.

```
topo_policy_file /etc/opensm/conf/topo_policy_file.cfg
```

Configuring a Topology Policy

The topologies policy file details a list of topologies. The policy file should be composed of one or more paragraphs which define a topology. Each paragraph should begin with the line 'topology' and end with the line 'end-topology'.

For example:

```
topology
…topology qualifiers…
end-topology
```

Topology Qualifiers

Unlike topology and end-topology which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. An inclusion of a column in the qualifier values will result in the policy's failure.

All topology qualifiers are mandatory. Absence of any of the below qualifiers will cause the policy parsing to fail.

Table 2 - Topology Qualifiers

Parameter	Description	Example
id	Topology ID. Legal Values – any positive value. Must be unique.	id: 1
sw-grp	Name of the port group that includes all switches and switch ports to be used in this topology.	sw-grp: ys_switches
hca-grp	Name of the port group that includes all HCA's to be used in this topology.	hca-grp: ys_hosts

Configuration File per Routing Engine

Each engine in the routing chain can be provided by its own configuration file. Routing engine configuration file is the fraction of parameters defined in the main opensm configuration file.

Some rules should be applied when defining a particular configuration file for a routing engine:

- Parameters that are not specified in specific routing engine configuration file are inherited from the main opensm configuration file.
- The following configuration parameters are taking effect only in the main opensm configuration file:
  - qos and qos_* settings like (vl_arb, sl2vl, etc.)
  - lmc
  - routing_engine

Defining a Routing Chain Policy File

In order to define a port group policy file, set the parameter 'rch_policy_file' in the opensm configuration file.

```
rch_policy_file /etc/opensm/conf/chains_policy_file
```

First Routing Engine in the Chain

The first unicast engine in a routing chain must include all switches and HCA's in the fabric (topology id must be 0). The path-bit parameter value is path-bit 0 and it cannot be changed.

Configuring a Routing Chains Policy

The routing chains policy file details the routing engines (and their fallback engines) used for the fabric's routing. The policy file should be composed of one or more paragraphs which defines an engine (or a fallback engine). Each paragraph should begin with the line 'unicast-step' and end with the line 'end-unicast-step'.

For example:

```
unicast-step
…routing engine qualifiers…
end-unicast-step
```

Routing Engine Qualifiers

Unlike unicast-step and end-unicast-step which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. An inclusion of a colon in the qualifier values will result in the policy's failure.

Parameter	Description	Example
id	'id' is man**dato**ry. Without an id qualifier for each engine, the policy fails. • Legal values – size_t value (0 is illegal). • The engines in the policy chain are set according to an ascending id order, so it is highly crucial to verify that the id that is given to the engines match the order in which you would like the engines to be set.	is: 1
engine	This is a mandatory qualifier that describes the routing algorithm used within this unicast step. Currently, on the first phase of routing chains, legal values are minhop/ftree/updn.	engine: minhop
use	This is an optional qualifier that enables one to describe the usage of this unicast step. If undefined, an empty string is used as a default.	use: ftree routing for for yellow stone nodes
config	This is an optional qualifier that enables one to define a separate opensm config file for a specific unicast step. If undefined, all parameters are taken from main opensm configuration file.	config: /etc/config/opensm2.cfg
topology	Define the topology that this engine uses. • Legal value – id of an existing topology that is defined in topologies policy (or zero that represents the entire fabric and not a specific topology). • Default value – If unspecified, a routing engine will relate to the entire fabric (as if topology zero was defined). • Notice: The first routing engine (the engine with the lowest id) MUST be configured with topology: 0 (entire fabric) or else, the routing chain parser will fail.	topology: 1

Parameter	Description	Example
fallback-to	This is an optional qualifier that enables one to define the current unicast step as a fallback to another unicast step. This can be done by defining the id of the unicast step that this step is a fallback to. • If undefined, the current unicast step is not a fallback. • If the value of this qualifier is a non-existent engine id, this step will be ignored. • A fallback step is meaningless if the step it is a fallback to did not fail. • It is impossible to define a fallback to a fall-back step (such definition will be ignored)	-
path-bit	This is an optional qualifier that enables one to define a specific lid offset to be used by the current unicast step. Setting lmc > 0 in main opensm configuration file is a prerequisite for assigning specific path-bit for the routing engine. Default value is 0 (if path-bit is not speci-fied)	Path-bit: 1

Dump Files per Routing Engine

Each routing engine on the chain will dump its own data files if the appropriate log_flags is set (for instance 0x43).

The files that are dumped by each engine are:

- opensm-lid-matrix.dump
- opensm-lfts.dump
- opensm.fdbs
- opensm-subnet.lst

These files should contain the relevant data for each engine topology.

sl2vl and mcfdbs files are dumped only once for the entire fabric and NOT by every rout-ing engine.

- Each engine concatenates its ID and routing algorithm name in its dump files names, as follows:
  - opensm-lid-matrix.2.minhop.dump
  - opensm.fdbs.3.ftree
  - opensm-subnet.4.updn.lst
- In case that a fallback routing engine is used, both the routing engine that failed and the fallback engine that replaces it, dump their data.

  If, for example, engine 2 runs ftree and it has a fallback engine with 3 as its id that runs minhop, one should expect to find 2 sets of dump files, one for each engine:

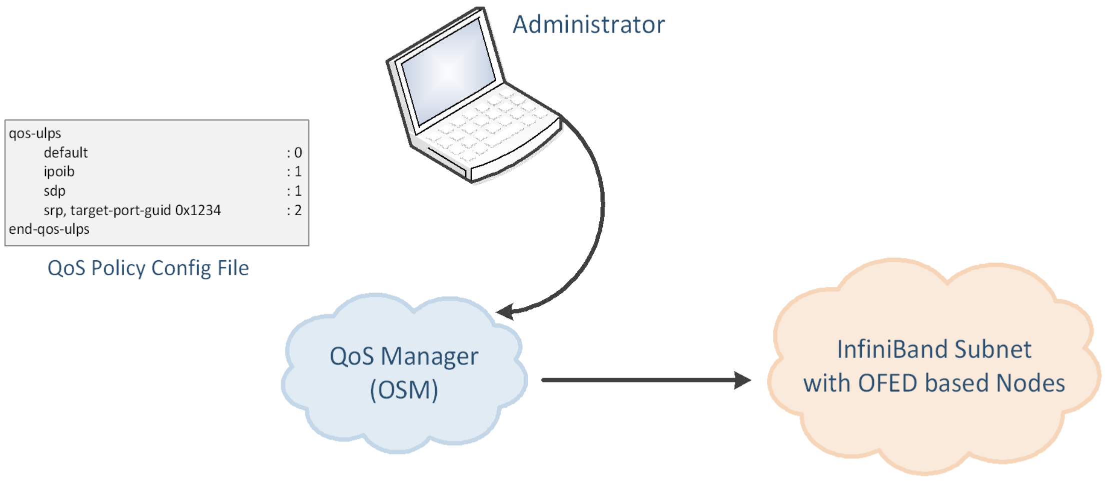
- opensm-lid-matrix.2.ftree.dump
- opensm-lid-matrix.3.minhop.dump
- opensm.fdbs.2.ftree
- opensm.fdbs.3.munhop

3.2.2.6 Quality of Service Management in OpenSM

When Quality of Service (QoS) in OpenSM is enabled (using the '-Q' or '--qos' flags), OpenSM looks for a QoS Policy file. During fabric initialization and at every heavy sweep, OpenSM parses the QoS policy file, applies its settings to the discovered fabric elements, and enforces the provided policy on client requests. The overall flow for such requests is as follows:

- The request is matched against the defined matching rules such that the QoS Level definition is found
- Given the QoS Level, a path(s) search is performed with the given restrictions imposed by that level

Figure 3: QoS Manager



```
qos-ulps
        default                                  : 0
        ipoib                                    : 1
        sdp                                      : 1
        srp, target-port-guid 0x1234             : 2
end-qos-ulps
```

There are two ways to define QoS policy:

- Advanced – the advanced policy file syntax provides the administrator various ways to match a PathRecord/MultiPathRecord (PR/MPR) request, and to enforce various QoS constraints on the requested PR/MPR
- Simple – the simple policy file syntax enables the administrator to match PR/MPR requests by various ULPs and applications running on top of these ULPs

Advanced QoS Policy File

The QoS policy file has the following sections:

I) Port Groups (denoted by port-groups)

This section defines zero or more port groups that can be referred later by matching rules (see below). Port group lists ports by:

- Port GUID
- Port name, which is a combination of NodeDescription and IB port number

- PKey, which means that all the ports in the subnet that belong to partition with a given PKey belong to this port group
- Partition name, which means that all the ports in the subnet that belong to partition with a given name belong to this port group
- Node type, where possible node types are: CA, SWITCH, ROUTER, ALL, and SELF (SM's port).

II) QoS Setup (denoted by qos-setup)

This section describes how to set up SL2VL and VL Arbitration tables on various nodes in the fabric. However, this is not supported in OFED. SL2VL and VLArb tables should be configured in the OpenSM options file (default location - /var/cache/opensm/opensm.opts).

III) QoS Levels (denoted by qos-levels)

Each QoS Level defines Service Level (SL) and a few optional fields:

- MTU limit
- Rate limit
- PKey
- Packet lifetime

When path(s) search is performed, it is done with regards to restriction that these QoS Level parameters impose. One QoS level that is mandatory to define is a DEFAULT QoS level. It is applied to a PR/MPR query that does not match any existing match rule. Similar to any other QoS Level, it can also be explicitly referred by any match rule.

IV) QoS Matching Rules (denoted by qos-match-rules)

Each PathRecord/MultiPathRecord query that OpenSM receives is matched against the set of matching rules. Rules are scanned in order of appearance in the QoS policy file such as the first match takes precedence.

Each rule has a name of QoS level that will be applied to the matching query. A default QoS level is applied to a query that did not match any rule.

Queries can be matched by:

- Source port group (whether a source port is a member of a specified group)
- Destination port group (same as above, only for destination port)
- PKey
- QoS class
- Service ID

To match a certain matching rule, PR/MPR query has to match ALL the rule's criteria. However, not all the fields of the PR/MPR query have to appear in the matching rule.

For instance, if the rule has a single criterion - Service ID, it will match any query that has this Service ID, disregarding rest of the query fields. However, if a certain query has only Service ID (which means that this is the only bit in the PR/MPR component mask that is on), it will not match any rule that has other matching criteria besides Service ID.

Simple QoS Policy Definition

Simple QoS policy definition comprises of a single section denoted by qos-ulps. Similar to the advanced QoS policy, it has a list of match rules and their QoS Level, but in this case a match rule has only one criterion - its goal is to match a certain ULP (or a certain application on top of this ULP) PR/MPR request, and QoS Level has only one constraint - Service Level (SL).

The simple policy section may appear in the policy file in combine with the advanced policy, or as a stand-alone policy definition. See more details and list of match rule criteria below.

Policy File Syntax Guidelines

- Leading and trailing blanks, as well as empty lines, are ignored, so the indentation in the example is just for better readability.
- Comments are started with the pound sign (#) and terminated by EOL.
- Any keyword should be the first non-blank in the line, unless it's a comment.
- Keywords that denote section/subsection start have matching closing keywords.
- Having a QoS Level named "DEFAULT" is a must - it is applied to PR/MPR requests that didn't match any of the matching rules.
- Any section/subsection of the policy file is optional.

Examples of Advanced Policy File

As mentioned earlier, any section of the policy file is optional, and the only mandatory part of the policy file is a default QoS Level.

Here's an example of the shortest policy file:

```
qos-levels
    qos-level
        name: DEFAULT
        sl: 0
    end-qos-level
end-qos-levels
```

Port groups section is missing because there are no match rules, which means that port groups are not referred anywhere, and there is no need defining them. And since this policy file doesn't have any matching rules, PR/MPR query will not match any rule, and OpenSM will enforce default QoS level. Essentially, the above example is equivalent to not having a QoS policy file at all.

The following example shows all the possible options and keywords in the policy file and their syntax:

```
#
# See the comments in the following example.
# They explain different keywords and their meaning.
#
port-groups

    port-group # using port GUIDs
        name: Storage
        # "use" is just a description that is used for logging
        #  Other than that, it is just a comment
        use: SRP Targets
        port-guid: 0x10000000000001, 0x10000000000005-0x1000000000FFFA
        port-guid: 0x1000000000FFFF
    end-port-group

    port-group
        name: Virtual Servers
        # The syntax of the port name is as follows:
        #   "node_description/Pnum".
        # node_description is compared to the NodeDescription of the node,
        # and "Pnum" is a port number on that node.
        port-name: vs1 HCA-1/P1, vs2 HCA-1/P1
    end-port-group

    # using partitions defined in the partition policy
    port-group
        name: Partitions
        partition: Part1
        pkey: 0x1234
    end-port-group

    # using node types: CA, ROUTER, SWITCH, SELF (for node that runs SM)
    # or ALL (for all the nodes in the subnet)
    port-group
        name: CAs and SM
        node-type: CA, SELF
    end-port-group

end-port-groups

qos-setup
    # This section of the policy file describes how to set up SL2VL and VL
    # Arbitration tables on various nodes in the fabric.
    # However, this is not supported in OFED - the section is parsed
    # and ignored. SL2VL and VLArb tables should be configured in the
    # OpenSM options file (by default - /var/cache/opensm/opensm.opts).
end-qos-setup

qos-levels

    # Having a QoS Level named "DEFAULT" is a must - it is applied to
    # PR/MPR requests that didn't match any of the matching rules.
    qos-level
        name: DEFAULT
        use: default QoS Level
        sl: 0
    end-qos-level

    # the whole set: SL, MTU-Limit, Rate-Limit, PKey, Packet Lifetime
    qos-level
        name: WholeSet
        sl: 1
        mtu-limit: 4
        rate-limit: 5
        pkey: 0x1234
        packet-life: 8
    end-qos-level

end-qos-levels

# Match rules are scanned in order of their apperance in the policy file.
# First matched rule takes precedence.
qos-match-rules

    # matching by single criteria: QoS class
    qos-match-rule
        use: by QoS class
        qos-class: 7-9,11
        # Name of qos-level to apply to the matching PR/MPR
        qos-level-name: WholeSet
    end-qos-match-rule

    # show matching by destination group and service id
    qos-match-rule
        use: Storage targets
        destination: Storage
        service-id: 0x10000000000001, 0x10000000000008-0x10000000000FFF
        qos-level-name: WholeSet
    end-qos-match-rule

    qos-match-rule
        source: Storage
        use: match by source group only
        qos-level-name: DEFAULT
    end-qos-match-rule

    qos-match-rule
        use: match by all parameters
        qos-class: 7-9,11
        source: Virtual Servers
        destination: Storage
```

```
            service-id: 0x0000000000010000-0x000000000001FFFF
            pkey: 0x0F00-0x0FFF
            qos-level-name: WholeSet
        end-qos-match-rule

    end-qos-match-rules
```

Simple QoS Policy - Details and Examples

Simple QoS policy match rules are tailored for matching ULPs (or some application on top of a ULP) PR/MPR requests. This section has a list of per-ULP (or per-application) match rules and the SL that should be enforced on the matched PR/MPR query.

Match rules include:

- Default match rule that is applied to PR/MPR query that didn't match any of the other match rules
- SDP
- SDP application with a specific target TCP/IP port range
- SRP with a specific target IB port GUID
- RDS
- IPoIB with a default PKey
- IPoIB with a specific PKey
- Any ULP/application with a specific Service ID in the PR/MPR query
- Any ULP/application with a specific PKey in the PR/MPR query
- Any ULP/application with a specific target IB port GUID in the PR/MPR query

Since any section of the policy file is optional, as long as basic rules of the file are kept (such as no referring to nonexisting port group, having default QoS Level, etc), the simple policy section (qos-ulps) can serve as a complete QoS policy file.

The shortest policy file in this case would be as follows:

```
qos-ulps
    default  : 0 #default SL
end-qos-ulps
```

It is equivalent to the previous example of the shortest policy file, and it is also equivalent to not having policy file at all. Below is an example of simple QoS policy with all the possible keywords:

```
qos-ulps
default              :0 # default SL
sdp, port-num 30000  :0 # SL for application running on
                                  # top of SDP when a destination
                                  # TCP/IPport is 30000
sdp, port-num 10000-20000     : 0
sdp                  :1 # default SL for any other
                                  # application running on top of SDP
rds                  :2 # SL for RDS traffic
ipoib, pkey 0x0001   :0 # SL for IPoIB on partition with
```

```
                                # pkey 0x0001
ipoib                :4 # default IPoIB partition,
                                # pkey=0x7FFF
any, service-id 0x6234:6 # match any PR/MPR query with a
                                # specific Service ID
any, pkey 0x0ABC      :6 # match any PR/MPR query with a
                                # specific PKey
srp, target-port-guid 0x1234  : 5 # SRP when SRP Target is located
                                # on a specified IB port GUID
any, target-port-guid 0x0ABC-0xFFFFF : 6 # match any PR/MPR query
                                # with a specific target port GUID
end-qos-ulps
```

Similar to the advanced policy definition, matching of PR/MPR queries is done in order of appearance in the QoS policy file such as the first match takes precedence, except for the "default" rule, which is applied only if the query didn't match any other rule. All other sections of the QoS policy file take precedence over the qos-ulps section. That is, if a policy file has both qos-match-rules and qos-ulps sections, then any query is matched first against the rules in the qos-match-rules section, and only if there was no match, the query is matched against the rules in qos-ulps section.

Note that some of these match rules may overlap, so in order to use the simple QoS definition effectively, it is important to understand how each of the ULPs is matched.

3.2.2.6.1 IPoIB

IPoIB query is matched by PKey or by destination GID, in which case this is the GID of the multicast group that OpenSM creates for each IPoIB partition.

Default PKey for IPoIB partition is 0x7fff, so the following three match rules are equivalent:

```
    ipoib      :<SL>
    ipoib, pkey 0x7fff : <SL>
    any,   pkey 0x7fff : <SL>
```

3.2.2.6.2 SDP

SDP PR query is matched by Service ID. The Service-ID for SDP is 0x000000000001PPPP, where PPPP are 4 hex digits holding the remote TCP/IP Port Number to connect to. The following two match rules are equivalent:

```
    sdp                                         :<SL>
    any, service-id 0x0000000000010000-0x000000000001ffff : <SL>
```

3.2.2.6.3 RDS

Similar to SDP, RDS PR query is matched by Service ID. The Service ID for RDS is 0x000000000106PPPP, where PPPP are 4 hex digits holding the remote TCP/IP Port Number to connect to. Default port number for RDS is 0x48CA, which makes a default Service-ID 0x00000000010648CA. The following two match rules are equivalent:

```
rds                        :<SL>
any, service-id 0x00000000010648CA : <SL>
```

3.2.2.6.4 SRP

Service ID for SRP varies from storage vendor to vendor, thus SRP query is matched by the target IB port GUID. The following two match rules are equivalent:

```
srp, target-port-guid 0x1234  : <SL>
any, target-port-guid 0x1234  : <SL>
```

Note that any of the above ULPs might contain target port GUID in the PR query, so in order for these queries not to be recognized by the QoS manager as SRP, the SRP match rule (or any match rule that refers to the target port guid only) should be placed at the end of the qos-ulps match rules.

3.2.2.6.5 MPI

SL for MPI is manually configured by MPI admin. OpenSM is not forcing any SL on the MPI traffic, and that's why it is the only ULP that did not appear in the qos-ulps section.

SL2VL Mapping and VL Arbitration

OpenSM cached options file has a set of QoS related configuration parameters, that are used to configure SL2VL mapping and VL arbitration on IB ports. These parameters are:

- Max VLs: the maximum number of VLs that will be on the subnet
- High limit: the limit of High Priority component of VL Arbitration table (IBA 7.6.9)
- VLArb low table: Low priority VL Arbitration table (IBA 7.6.9) template
- VLArb high table: High priority VL Arbitration table (IBA 7.6.9) template
- SL2VL: SL2VL Mapping table (IBA 7.6.6) template. It is a list of VLs corresponding to SLs 0-15 (Note that VL15 used here means drop this SL).

There are separate QoS configuration parameters sets for various target types: CAs, routers, switch external ports, and switch's enhanced port 0. The names of such parameters are prefixed by "qos_<type>_" string. Here is a full list of the currently supported sets:

- qos_ca_ - QoS configuration parameters set for CAs.
- qos_rtr_ - parameters set for routers.
- qos_sw0_ - parameters set for switches' port 0.
- qos_swe_ - parameters set for switches' external ports.

Here's the example of typical default values for CAs and switches' external ports (hard-coded in OpenSM initialization):

```
qos_ca_max_vls 15
qos_ca_high_limit 0
qos_ca_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_ca_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 0
qos_swe_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_swe_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

VL arbitration tables (both high and low) are lists of VL/Weight pairs. Each list entry contains a VL number (values from 0-14), and a weighting value (values 0-255), indicating the number of 64 byte units (credits) which may be transmitted from that VL when its turn in the arbitration occurs. A weight of 0 indicates that this entry should be skipped. If a list entry is programmed for VL15 or for a VL that is not supported or is not currently configured by the port, the port may either skip that entry or send from any supported VL for that entry.

Note, that the same VLs may be listed multiple times in the High or Low priority arbitration tables, and, further, it can be listed in both tables. The limit of high-priority VLArb table (qos_<type>_high_limit) indicates the number of high-priority packets that can be transmitted without an opportunity to send a low-priority packet. Specifically, the number of bytes that can be sent is high_limit times 4K bytes.

A high_limit value of 255 indicates that the byte limit is unbounded.

If the 255 value is used, the low priority VLs may be starved.

A value of 0 indicates that only a single packet from the high-priority table may be sent before an opportunity is given to the low-priority table.

Keep in mind that ports usually transmit packets of size equal to MTU. For instance, for 4KB MTU a single packet will require 64 credits, so in order to achieve effective VL arbitration for packets of 4KB MTU, the weighting values for each VL should be multiples of 64.

Below is an example of SL2VL and VL Arbitration configuration on subnet:

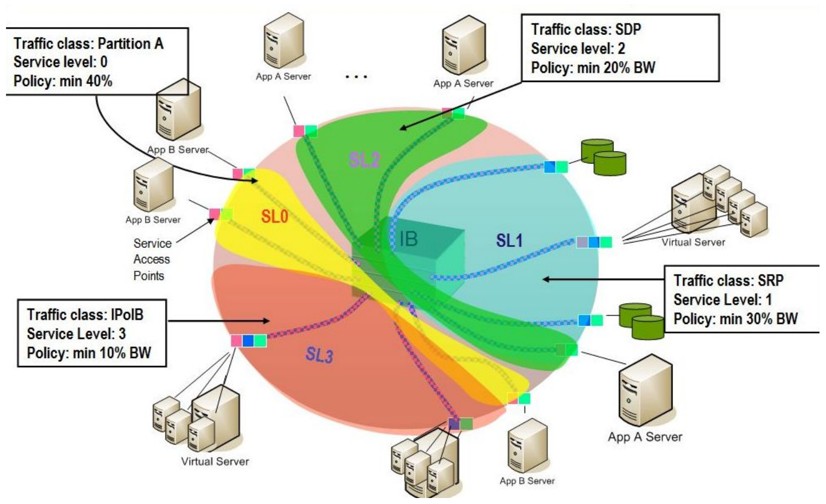
```
qos_ca_max_vls 15
qos_ca_high_limit 6
qos_ca_vlarb_high 0:4
qos_ca_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 6
qos_swe_vlarb_high 0:4
qos_swe_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

In this example, there are 8 VLs configured on subnet: VL0 to VL7. VL0 is defined as a high priority VL, and it is limited to 6 x 4KB = 24KB in a single transmission burst. Such configuration would suilt VL that needs low latency and uses small MTU when transmitting packets. Rest of VLs are defined as low priority VLs with different weights, while VL4 is effectively turned off.

Deployment Example

Figure 4 shows an example of an InfiniBand subnet that has been configured by a QoS manager to provide different service levels for various ULPs.

Figure 4: Example QoS Deployment on InfiniBand Subnet



3.2.2.7 QoS Configuration Examples

The following are examples of QoS configuration for different cluster deployments. Each example provides the QoS level assignment and their administration via OpenSM configuration files.

Typical HPC Example: MPI and Lustre

Assignment of QoS Levels

- MPI
  - Separate from I/O load
  - Min BW of 70%
- Storage Control (Lustre MDS)
  - Low latency
- Storage Data (Lustre OST)
  - Min BW 30%

Administration

- MPI is assigned an SL via the command line

```
host1# mpirun –sl 0
```

- OpenSM QoS policy file

In the following policy file example, replace OST* and MDS* with the real port GUIDs.

```
qos-ulps
```

```
    default                                                        :0 # default SL (for
MPI)
    any, target-port-guid OST1,OST2,OST3,OST4:1 # SL for Lustre OST
    any, target-port-guid MDS1,MDS2                                :2 # SL for Lustre
MDS
end-qos-ulps
```

- OpenSM options file

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 2:1
qos_vlarb_low 0:96,1:224
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

EDC SOA (2-tier): IPoIB and SRP

The following is an example of QoS configuration for a typical enterprise data center (EDC) with service oriented architecture (SOA), with IPoIB carrying all application traffic and SRP used for storage.

QoS Levels

- Application traffic
  - IPoIB (UD and CM) and SDP
  - Isolated from storage
  - Min BW of 50%
- SRP
  - Min BW 50%
  - Bottleneck at storage nodes

Administration

- OpenSM QoS policy file

In the following policy file example, replace SRPT* with the real SRP Target port GUIDs.

```
qos-ulps
    default                                  :0
    ipoib                                    :1
    sdp                                      :1
    srp, target-port-guid SRPT1,SRPT2,SRPT3 :2
end-qos-ulps
```

- OpenSM options file

```
qos_max_vls 8
 qos_high_limit 0
 qos_vlarb_high 1:32,2:32
 qos_vlarb_low 0:1,
 qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

EDC (3-tier): IPoIB, RDS, SRP

The following is an example of QoS configuration for an enterprise data center (EDC), with IPoIB carrying all application traffic, RDS for database traffic, and SRP used for storage.

QoS Levels

- Management traffic (ssh)
  - IPoIB management VLAN (partition A)
  - Min BW 10%
- Application traffic
  - IPoIB application VLAN (partition B)
  - Isolated from storage and database
  - Min BW of 30%
- Database Cluster traffic
  - RDS
  - Min BW of 30%
- SRP
  - Min BW 30%
  - Bottleneck at storage nodes

Administration

- OpenSM QoS policy file

In the following policy file example, replace SRPT* with the real SRP Initiator port GUIDs.

```
qos-ulps
    default                            :0
    ipoib, pkey 0x8001                 :1
    ipoib, pkey 0x8002                 :2
    rds                                :3
    srp, target-port-guid SRPT1, SRPT2, SRPT3  : 4
end-qos-ulps
```

- OpenSM options file

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 1:32,2:96,3:96,4:96
qos_vlarb_low 0:1
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

- Partition configuration file

```
Default=0x7fff,     ipoib : ALL=full;
PartA=0x8001, sl=1, ipoib : ALL=full;
```

Adaptive Routing

Adaptive Routing is at beta stage.

Adaptive Routing (AR) enables the switch to select the output port based on the port's load. AR supports two routing modes:

- Free AR: No constraints on output port selection.
- Bounded AR: The switch does not change the output port during the same transmission burst. This mode minimizes the appearance of out-of-order packets.

Adaptive Routing Manager enables and configures Adaptive Routing mechanism on fabric switches. It scans all the fabric switches, deduces which switches support Adaptive Routing and configures the AR functionality on these switches.

Currently, Adaptive Routing Manager supports only link aggregation algorithm. Adaptive Routing Manager configures AR mechanism to allow switches to select output port out of all the ports that are linked to the same remote switch. This algorithm suits any topology with several links between switches. Especially, it suits 3D torus/mesh, where there are several link in each direction of the X/Y/Z axis.

If some switches do not support AR, they will slow down the AR Manager as it may get timeouts on the AR-related queries to these switches.

Installing the Adaptive Routing

Adaptive Routing Manager is a Subnet Manager plug-in, i.e. it is a shared library (libarmgr.so) that is dynamically loaded by the Subnet Manager. Adaptive Routing Manager is installed as a part of Mellanox OFED installation.

Running Subnet Manager with Adaptive Routing Manager

Adaptive Routing (AR) Manager can be enabled/disabled through SM options file.

3.2.2.7.1 Enabling Adaptive Routing

To enable Adaptive Routing, perform the following:

Step 1. Create the Subnet Manager options file. Run:

```
opensm -c <options-file-name>
```

Step 2. Add '**armgr**' to the '**event_plugin_name**' option in the file:

```
# Event plugin name(s)
event_plugin_name armgr
```

Step 3. Run Subnet Manager with the new options file:

```
opensm -F <options-file-name>
```

Adaptive Routig Manager can read options file with various configuration parameters to fine-tune AR mechanism and AR Manager behavior. Default location of the AR Manager options file is **/etc/opensm/ar_mgr.conf**.

To provide an alternative location, please perform the following:

Step 1. Add '**armgr --conf_file <ar-mgr-options-file-name>**' to the '**event_plugin_options**' option in the file **# Options** string that would be passed to the plugin(s) **event_plugin_options armgr --conf_file <ar-mgr-options-file-name>**

Step 2. Run Subnet Manager with the new options file:

```
opensm -F <options-file-name>
```

See an example of AR Manager options file with all the default values in "Example of Adaptive Routing Manager Options File" on page 130.

3.2.2.7.2 Disabling Adaptive Routing

There are two ways to disable Adaptive Routing Manager:

1. By disabling it explicitly in the Adaptive Routing configuration file.
2. By removing the 'armgr' option from the Subnet Manager options file.

Adaptive Routing mechanism is automatically disabled once the switch receives setting of the usual linear routing table (LFT).

Therefore, no action is required to clear Adaptive Routing configuration on the switches if you do not wish to use Adaptive Routing.

Querying Adaptive Routing Tables

When Adaptive Routing is active, the content of the usual Linear Forwarding Routing Table on the switch is invalid, thus the standard tools that query LFT (e.g. smpquery, dump_lfts.sh, and others) cannot be used. To query the switch for the content of its Adaptive Routing table, use the 'smparquery' tool that is installed as a part of the Adaptive Routing Manager package. To see its usage details, run 'smparquery -h'.

Adaptive Routing Manager Options File

The default location of the AR Manager options file is /etc/opensm/ar_mgr.conf. To set an alternative location, please perform the following:

1. Add 'armgr --conf_file **`<ar-mgr-options-file-name>`** to the **`event_plugin_option`**' option in the file **`# Options`** string that would be passed to the plugin(s) **`event_plugin_options armgr --conf_file <ar-mgr-options-file-name>`**
2. Run Subnet Manager with the new options file:

```
opensm -F <options-file-name>'
```

AR Manager options file contains two types of parameters:

1. General options - Options which describe the AR Manager behavior and the AR parameters that will be applied to all the switches in the fabric.
2. Per-switch options - Options which describe specific switch behavior.

Note the following:

- Adaptive Routing configuration file is case sensitive.
- You can specify options for nonexisting switch GUID. These options will be ignored until a switch with a matching GUID will be added to the fabric.
- Adaptive Routing configuration file is parsed every AR Manager cycle, which in turn is executed at every heavy sweep of the Subnet Manager.
- If the AR Manager fails to parse the options file, default settings for all the options will be used.

3.2.2.7.3 General AR Manager Options

Option File	Description	Values
ENABLE: <true|false>	Enable/disable Adaptive Routing on fabric switches. Note that if a switch was identified by AR Manager as device that does not support AR, AR Manager will not try to enable AR on this switch. If the firmware of this switch was updated to support the AR, the AR Manager will need to be restarted (by restarting Subnet Manager) to allow it to configure the AR on this switch. This option can be changed on-the-fly.	Default: true
AR_ALGORITHM: <LAG|TREE>	Adaptive Routing algorithm: • LAG: Ports groups are created out of "parallel" links. Links that are connecting the same pair of switches. • TREE: All the ports with minimal hops to destination are in the same group. Must run together with UPDN routing engine.	
AR_MODE: <bounded|free>	Adaptive Routing Mode: • free: no constraints on output port selection • bounded: the switch does not change the output port during the same transmission burst. This mode minimizes the appearance of out-of-order packets This option can be changed on-the-fly.	Default: bounded
AGEING_TIME: <usec>	Applicable to bounded AR mode only. Specifies how much time there should be no traffic in order for the switch to declare a transmission burst as finished and allow changing the output port for the next transmission burst (32-bit value). This option can be changed on-the-fly.	Default: 30
MAX_ERRORS: <N> ERROR_WINDOW: <N>	When number of errors exceeds 'MAX_ERRORS' of send/receive errors or timeouts in less than 'ERROR_WINDOW' seconds, the AR Manager will abort, returning control back to the Subnet Manager. This option can be changed on-the-fly.	Values for both options: [0-0xffff] • MAX_ERRORS = 0: zero tollerance - abort configuration on first error. Default:10 • ERROR_WINDOW = 0: mechanism disabled - no error checking. Default: 5
LOG_FILE: <full path>	AR Manager log file. This option can be changed on-the-fly.	Default: /var/log/armgr.log

Option File	Description	Values
LOG_SIZE: <size in MB>	This option defines maximal AR Manager log file size in MB. The logfile will be truncated and restarted upon reaching this limit. This option cannot be changed on-the-fly.	0: unlimited log file size. Default: 5

Per-switch AR Options

A user can provide per-switch configuration options with the following syntax:

```
SWITCH <GUID> {
                  <switch option 1>;
                  <switch option 2>;
                  ...
                  }
```

The following are the per-switch options:

Option File	Description	Values
ENABLE: <true|false>	Allows you to enable/disable the AR on this switch. If the general ENABLE option value is set to 'false', then this per-switch option is ignored. This option can be changed on the fly.	Default: true
AGEING_TIME: <usec>	Applicable to bounded AR mode only. Specifies how much time there should be no traffic in order for the switch to declare a transmission burst as finished and allow changing the output port for the next transmission burst (32-bit value). In the pre-switch options file this option refers to the particular switch only This option can be changed on-the-fly.	Default: 30

Example of Adaptive Routing Manager Options File

```
ENABLE: true;
 LOG_FILE: /tmp/ar_mgr.log;
 LOG_SIZE: 100;
 MAX_ERRORS: 10;
 ERROR_WINDOW: 5;

SWITCH 0x12345 {
ENABLE: true;
AGEING_TIME: 77;
 }

SWITCH 0x0002c902004050f8 {
```

```
AGEING_TIME: 44;
 }

SWITCH 0xabcde {
ENABLE: false;
 }
```

3.2.2.8 Congestion Control

Congestion Control Manager is a Subnet Manager (SM) plug-in, i.e. it is a shared library (libccmgr.so) that is dynamically loaded by the Subnet Manager. Congestion Control Manager is installed as part of Mellanox OFED installation.

The Congestion Control mechanism controls traffic entry into a network and attempts to avoid over-subscription of any of the processing or link capabilities of the intermediate nodes and networks. Additionally, is takes resource reducing steps by reducing the rate of sending packets. Congestion Control Manager enables and configures Congestion Control mechanism on fabric nodes (HCAs and switches).

Running OpenSM with Congestion Control Manager

Congestion Control (CC) Manager can be enabled/disabled through SM options file. To do so, perform the following:

Step 1. Create the file. Run:

```
opensm -c <options-file-name>'
```

Step 2. Find the '**`event_plugin_name`**' option in the file, and add '**`ccmgr`**' to it.

```
# Event plugin name(s)
event_plugin_name ccmgr
```

Step 3. Run the SM with the new options file: '**`opensm -F <options-file-name>`**'

Once the Congestion Control is enabled on the fabric nodes, to completely disable Congestion Control, you will need to actively turn it off. Running the SM w/o the CC Manager is not sufficient, as the hardware still continues to function in accordance to the previous CC configuration.

For further information on how to turn OFF CC, please refer to Section , “Configuring Congestion Control Manager”, on page 131

Configuring Congestion Control Manager

Congestion Control (CC) Manager comes with a predefined set of setting. However, you can fine-tune the CC mechanism and CC Manager behavior by modifying some of the options. To do so, perform the following:

Step 1. Find the '**`event_plugin_options`**' option in the SM options file, and add the following:

```
conf_file <cc-mgr-options-file-name>':
# Options string that would be passed to the plugin(s)
event_plugin_options ccmgr --conf_file <cc-mgr-options-file-name>
```

Step 2. Run the SM with the new options file: '**`opensm -F <options-file-name>'`**

To turn CC OFF, set 'enable' to 'FALSE' in the Congestion Control Manager configuration file, and run OpenSM ones with this configuration.

For the full list of CC Manager options with all the default values, See "Configuring Congestion Control Manager" on page 131.

For further details on the list of CC Manager options, please refer to the IB spec.

Configuring Congestion Control Manager Main Settings

To fine-tune CC mechanism and CC Manager behavior, and set the CC manager main settings, perform the following:

- To enables/disables Congestion Control mechanism on the fabric nodes, set the following parameter:

  ```
  enable
  ```

  - The values are: **<TRUE | FALSE>**.
  - The default is: **True**
- CC manager configures CC mechanism behavior based on the fabric size. The larger the fabric is, the more aggressive CC mechanism is in its response to congestion. To manually modify CC manager behavior by providing it with an arbitrary fabric size, set the following parameter:

  ```
  num_hosts
  ```

  - The values are: **[0-48K]**.
  - The default is: **0** (base on the CCT calculation on the current subnet size)
- The smaller the number value of the parameter, the faster HCAs will respond to the congestion and will throttle the traffic. Note that if the number is too low, it will result in suboptimal bandwidth. To change the mean number of packets between marking eligible packets with a FECN, set the following parameter:

  ```
  marking_rate
  ```

  - The values are: **[0-0xffff]**.
  - The default is: **0xa**
- You can set the minimal packet size that can be marked with FECN. Any packet less than this size [bytes] will not be marked with FECN. To do so, set the following parameter:

  ```
  packet_size
  ```

  - The values are: **[0-0x3fc0]**.
  - The default is: **0x200**
- When number of errors exceeds 'max_errors' of send/receive errors or timeouts in less than 'error_window' seconds, the CC MGR will abort and will allow OpenSM to proceed. To do so, set the following parameter:

  ```
  max_errors
  error_window
  ```

- The values are:

```
max_errors = 0: zero tollerance - abort configuration on first error
error_window = 0: mechanism disabled - no error checking.[0-48K]
```

- The default is: `5`

3.2.2.8.1 Congestion Control Manager Options File

Option File	Description	Values
enable	Enables/disables Congestion Control mechanism on the fabric nodes.	Values: <TRUE | FALSE> Default: True
num_hosts	Indicates the number of nodes. The CC table values are calculated based on this number.	Values: [0-48K] Default: 0 (base on the CCT calculation on the current subnet size)
threshold	Indicates how aggressive the congestion marking should be.	[0-0xf] • 0 - no packet marking, • 0xf - very aggressive Default: 0xf
marking_rate	The mean number of packets between marking eligible packets with a FECN	Values: [0-0xffff] Default: 0xa
packet_size	Any packet less than this size [bytes] will not be marked with FECN.	Values: [0-0x3fc0] Default: 0x200
port_control	Specifies the Congestion Control attribute for this port	Values: • 0 - QP based congestion control, • 1 - SL/Port based congestion control Default: 0
ca_control_map	An array of sixteen bits, one for each SL. Each bit indicates whether or not the corresponding SL entry is to be modified.	Values: 0xffff
ccti_increase	Sets the CC Table Index (CCTI) increase.	Default: 1
trigger_threshold	Sets the trigger threshold.	Default: 2
ccti_min	Sets the CC Table Index (CCTI) minimum.	Default: 0
cct	Sets all the CC table entries to a specified value . The first entry will remain 0, whereas last value will be set to the rest of the table.	Values: <comma-separated list> Default: 0 When the value is set to 0, the CCT calculation is based on the number of nodes.

Option File	Description	Values
ccti_timer	Sets for all SL's the given ccti timer.	Default: 0 When the value is set to 0, the CCT calculation is based on the number of nodes.
max_errors error_window	When number of errors exceeds 'max_errors' of send/receive errors or timeouts in less than 'error_window' seconds, the CC MGR will abort and will allow OpenSM to proceed.	Values: • max_errors = 0: zero tolle-rance - abort configuration on first error. • error_window = 0: mecha-nism disabled - no error checking. Default: 5
cc_statistics_-cycle	Enables CC MGR to collect statistics from all nodes every cc_statistics_cycle [seconds]	Default: 0 When the value is set to 0, no statistics are collected.

3.2.2.9 DOS MAD Prevention

DOS MAD prevention is achieved by assigning a threshold for each agent's RX. Agent's RX threshold provides a protection mechanism to the host memory by limiting the agents' RX with a threshold. Incoming MADs above the threshold are dropped and are not queued to the agent's RX.

The default threshold value is 10,000 for non-manager class agents and 100,000 for manager class agents (SM for e.g.). By default the feature is disabled.

➢ ***To enable DOS MAD Prevention:***

Step 1. Go to /etc/modprobe.d/mlnx.conf.

Step 2. Add to the file the option below.

```
ib_umad enable_rx_threshold 1
```

The threshold's value can be controlled from the user-space via libibumad.

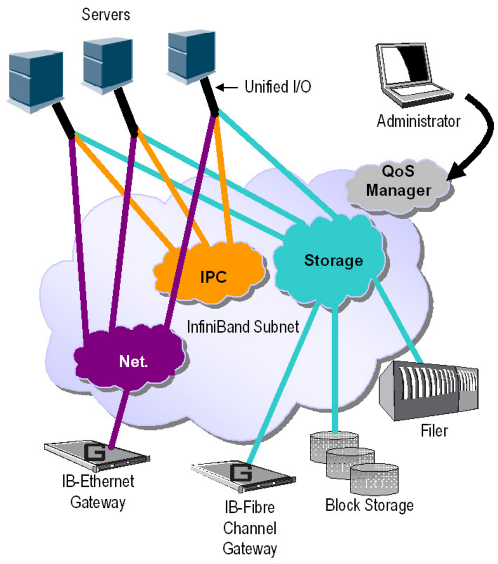
To change the value, use the following API:

```
int umad_update_threshold(int fd, int threshold);

@fd: file descriptor, agent's RX associated to this fd.
@threshold: new threshold value
```

3.2.3 Quality of Service (QoS)

Quality of Service (QoS) requirements stem from the realization of I/O consolidation over an IB network. As multiple applications and ULPs share the same fabric, a means is needed to control their use of network resources.

Figure 5: I/O Consolidation Over InfiniBand



The basic need is to differentiate the service levels provided to different traffic flows, such that a policy can be enforced and can control each flow utilization of fabric resources.

The InfiniBand Architecture Specification defines several hardware features and management interfaces for supporting QoS:

- Up to 15 Virtual Lanes (VL) carry traffic in a non-blocking manner
- Arbitration between traffic of different VLs is performed by a two-priority-level weighted round robin arbiter. The arbiter is programmable with a sequence of (VL, weight) pairs and a maximal number of high priority credits to be processed before low priority is served
- Packets carry class of service marking in the range 0 to 15 in their header SL field
- Each switch can map the incoming packet by its SL to a particular output VL, based on a programmable table VL=SL-to-VL-MAP(in-port, out-port, SL)
- The Subnet Administrator controls the parameters of each communication flow by providing them as a response to Path Record (PR) or MultiPathRecord (MPR) queries

DiffServ architecture (IETF RFC 2474 & 2475) is widely used in highly dynamic fabrics. The following subsections provide the functional definition of the various software elements that enable a DiffServ-like architecture over the Mellanox OFED software stack.

3.2.3.1 QoS Architecture

QoS functionality is split between the SM/SA, CMA and the various ULPs. We take the “chronology approach” to describe how the overall system works.

1. The network manager (human) provides a set of rules (policy) that define how the network is being configured and how its resources are split to different QoS-Levels. The policy also define how to decide which QoS-Level each application or ULP or service use.
2. The SM analyzes the provided policy to see if it is realizable and performs the necessary fabric setup. Part of this policy defines the default QoS-Level of each partition. The SA is enhanced to match the requested Source, Destination, QoS-Class, Service-ID, PKey against the policy, so clients (ULPs, programs) can obtain a policy enforced QoS. The SM may also set up partitions with appropriate IPoIB broadcast group. This broadcast group carries its QoS attributes: SL, MTU, RATE, and Packet Lifetime.
3. IPoIB is being setup. IPoIB uses the SL, MTU, RATE and Packet Lifetime available on the multicast group which forms the broadcast group of this partition.
4. MPI which provides non IB based connection management should be configured to run using hard coded SLs. It uses these SLs for every QP being opened.
5. ULPs that use CM interface (like SRP) have their own pre-assigned Service-ID and use it while obtaining PathRecord/MultiPathRecord (PR/MPR) for establishing connections. The SA receiving the PR/MPR matches it against the policy and returns the appropriate PR/MPR including SL, MTU, RATE and Lifetime.
6. ULPs and programs (e.g. SDP) use CMA to establish RC connection provide the CMA the target IP and port number. ULPs might also provide QoS-Class. The CMA then creates Service-ID for the ULP and passes this ID and optional QoS-Class in the PR/MPR request. The resulting PR/MPR is used for configuring the connection QP.

PathRecord and MultiPathRecord Enhancement for QoS:

As mentioned above, the PathRecord and MultiPathRecord attributes are enhanced to carry the Service-ID which is a 64bit value. A new field QoS-Class is also provided.

A new capability bit describes the SM QoS support in the SA class port info. This approach provides an easy migration path for existing access layer and ULPs by not introducing new set of PR/MPR attributes.

3.2.3.2 Supported Policy

The QoS policy, which is specified in a stand-alone file, is divided into the following four subsections:

Port Group

A set of CAs, Routers or Switches that share the same settings. A port group might be a partition defined by the partition manager policy, list of GUIDs, or list of port names based on NodeDescription.

Fabric Setup

Defines how the SL2VL and VLArb tables should be setup.

In OFED this part of the policy is ignored. SL2VL and VLArb tables should be configured in the OpenSM options file (opensm.opts).

QoS-Levels Definition

This section defines the possible sets of parameters for QoS that a client might be mapped to. Each set holds SL and optionally: Max MTU, Max Rate, Packet Lifetime and Path Bits.

Path Bits are not implemented in OFED.

Matching Rules

A list of rules that match an incoming PR/MPR request to a QoS-Level. The rules are processed in order such as the first match is applied. Each rule is built out of a set of match expressions which should all match for the rule to apply. The matching expressions are defined for the following fields:

- SRC and DST to lists of port groups
- Service-ID to a list of Service-ID values or ranges
- QoS-Class to a list of QoS-Class values or ranges

3.2.3.3 CMA Features

The CMA interface supports Service-ID through the notion of port space as a prefix to the port number, which is part of the sockaddr provided to rdma_resolve_add(). The CMA also allows the ULP (like SDP) to propagate a request for a specific QoS-Class. The CMA uses the provided QoS-Class and Service-ID in the sent PR/MPR.

IPoIB

IPoIB queries the SA for its broadcast group information and uses the SL, MTU, RATE and Packet Lifetime available on the multicast group which forms this broadcast group.

SRP

The current SRP implementation uses its own CM callbacks (not CMA). So SRP fills in the Service-ID in the PR/MPR by itself and use that information in setting up the QP.

SRP Service-ID is defined by the SRP target I/O Controller (it also complies with IBTA Service-ID rules). The Service-ID is reported by the I/O Controller in the ServiceEntries DMA attribute and should be used in the PR/MPR if the SA reports its ability to handle QoS PR/MPRs.

3.2.3.4 OpenSM Features

The InfiniBand Subnet Manager (SM) be split into two main parts:

I. Fabric Setup

During fabric initialization, the Subnet Manager parses the policy and apply its settings to the discovered fabric elements.

II. PR/MPR Query Handling

OpenSM enforces the provided policy on client request. The overall flow for such requests is: first the request is matched against the defined match rules such that the target QoS-Level definition is found. Given the QoS-Level a path(s) search is performed with the given restrictions imposed by that level.

3.2.4 Secure Host

Secure host (supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only) enables the device to protect itself and the subnet from malicious software. This is achieved by:

- Not allowing untrusted entities to access the device configuration registers, directly (through pci_cr or pci_conf) and indirectly (through MADs)
- Hiding the M_Key from the untrusted entities
- Preventing the modification of GUID0 by the untrusted entities
- Preventing drivers on untrusted hosts to receive or transmit SMP (QP0) MAD packets (SMP firewall)

When the SMP firewall is enabled, the firmware handles all QP0 packets, and does not forward them to the driver. Any information required by the driver for proper operation (e.g., SM lid) is passed via events generated by the firmware while processing QP0 MADs.

Driver support mainly requires using the MAD_DEMUX firmware command at driver startup.

3.2.4.1 Secure Mode Operation

Secure mode capability is enabled by setting the "cr_protection_en" parameter set to 1 in the [HCA] section of the .ini file and then burning the firmware with this .ini file. If the parameter is set to zero, or is missing, secure-mode operation will not be possible.

Once the firmware allows secure-mode operation, the secure-mode capability must be activated by using "flint" to set a 64-bit key (and then restarting the driver).

The flint command is as follows (the key is specified as up to 16 hex digits):

```
flint -d <device> set_key <64-bit key>
```

Example:

```
flint -d /dev/mst/mt26428_pci_cr0 set_key 1a1a1a1a2b2b2b2b
```

3.2.4.1.1 Enabling/Disabling Hardware Access

Once a 64-bit key is installed, the secure-mode is active once the driver is restarted. If the host is rebooted, the HCA comes out of reboot with secure-mode enabled. Hardware access can be disabled while the driver is running to enable operation such as maintenance, or firmware burning and then restored at the end of the operation.

The temporary enable does not affect the SMP firewall. This remains active even if the "cr-space" is temporarily permitted.

➢ ***To enable hardware access:***

```
flint -d /dev/mst/mt26428_pci_cr0 hw_access enable
Enter Key: ********
```

➢ ***To disable hardware access:***

```
flint -d /dev/mst/mt26428_pci_cr0 hw_access disable
```

If you do not explicitly restore hardware access when the maintenance operation is completed, the driver restart will NOT do so. The driver will come back after restart with hardware access disabled. Note, though, that the SMP firewall will still be active.

A host reboot will restore hardware access (with SMP firewall active). Thus, when you disable hardware access, you should restore it immediately after maintenance has been completed, either by using the flint command above or by rebooting the host (or both).

3.2.4.1.2 Burning New Firmware when Secure-Mode is Active

➢ ***To burn a new firmware when the secure-mode is active:***

Step 1. Temporarily enable the hardware access (see section Enabling/Disabling hardware access).

Step 2. Burn the new firmware.

Step 3. Reboot the host (not just restart the driver).

3.2.4.1.3 Permanently Disabling Secure-Mode

➢ ***To permanently disabled Secure-mode by setting the pass-key to zero:***

Step 1. Temporarily disable the secure-mode (see section "Enabling/Disabling Hardware Access" on page 139).

Step 2. Reset the pass-key to zero.

```
flint -d <device> set_key 0
```

Step 3. Reboot the host.

This operation will cause the HCA to always come up (even from host reboot) in unsecured mode. To restore security, simply set a non-zero pass-key again.

3.2.4.1.4 Checking if Hardware Access is Active

➢ ***To check if hardware access is active:***

```
flint -d /dev/mst/mt26428_pci_cr0 -qq q
```

If the hardware access is active, you will see the following error message:

```
E- Cannot open /dev/mst/mt26428_pci_cr0: HW access is disabled on the device.
E- Run "flint -d /dev/mst/mt26428_pci_cr0 hw_access enable" in order to enable HW access.
```

3.2.4.1.5 Checking if the SMP Firewall is Active

The SMP firewall is active as long as there is a non-zero pass-key active in the firmware (regardless of whether or not the Secure-Mode has been temporarily disabled).

To check if SMP Firewall is active, run the InfiniBand diagnostic command sminfo.

If the SMP firewall is active, the command will fail as shown below:

```
[root@dev-l-vrt-016 ~]# sminfo
ibwarn: [26872] mad_rpc: _do_madrpc failed; dport (Lid 1)
sminfo: iberror: failed: query
```

3.2.5 Upper Layer Protocols

3.2.5.1 IP over InfiniBand (IPoIB)

The IP over IB (IPoIB) driver is a network interface implementation over InfiniBand. IPoIB encapsulates IP datagrams over an InfiniBand Connected or Datagram transport service. The IPoIB driver, ib_ipoib, exploits the following capabilities:

- VLAN simulation over an InfiniBand network via child interfaces
- High Availability via Bonding
- Varies MTU values:
  - up to 4k in Datagram mode
  - up to 64k in Connected mode
- Uses any ConnectX® IB ports (one or two)
- Inserts IP/UDP/TCP checksum on outgoing packets
- Calculates checksum on received packets
- Support net device TSO through ConnectX® LSO capability to defragment large datagrams to MTU quantas.
- Dual operation mode - datagram and connected
- Large MTU support through connected mode

IPoIB also supports the following software based enhancements:

- Giant Receive Offload
- NAPI
- Ethtool support

3.2.5.1.1 IPoIB Mode Setting

IPoIB can run in two modes of operation: Connected mode and Datagram mode. By default, IPoIB is set to work in Datagram, except for Connect-IB® adapter card which uses IPoIB with Connected mode as default.

For better scalability and performance we recommend using the Datagram mode. However, the mode can be changed to Connected mode by editing the file `/etc/infiniband/openib.conf` and setting `'SET_IPOIB_CM=yes'`.

The `SET_IPOIB_CM` parameter is set to `"auto"` by default to enable the Connected mode for Connect-IB® card and Datagram for all other ConnectX® cards.

After changing the mode, you need to restart the driver by running:

```
/etc/init.d/openibd restart
```

To check the current mode used for out-going connections, enter:

```
cat /sys/class/net/ib<n>/mode
```

Changing the IPoIB mode (CM vs UD) requires the interface to be in 'down' state.

3.2.5.1.1.1 Port Configuration

The physical port MTU in Datagram mode (indicates the port capability) default value is 4k, whereas the IPoIB port MTU ("logical" MTU) default value is 2k as it is set by the OpenSM.

To change the IPoIB MTU to 4k, edit the OpenSM partition file in the section of IPoIB setting as follow:

```
Default=0xffff, ipoib, mtu=5 : ALL=full;
```

*Where "mtu=5" indicates that all IPoIB ports in the fabric are using 4k MTU, ("mtu=4" indicates 2k MTU)

3.2.5.1.2 IPoIB Configuration

Unless you have run the installation script `mlnxofedinstall` with the flag '-n', then IPoIB has not been configured by the installation. The configuration of IPoIB requires assigning an IP address and a subnet mask to each HCA port, like any other network adapter card (i.e., you need to prepare a file called ifcfg-ib<n> for each port). The first port on the first HCA in the host is called interface ib0, the second port is called ib1, and so on.

An IPoIB configuration can be based on DHCP (Section 3.2.5.1.2.1) or on a static configuration (Section 3.2.5.1.2.4) that you need to supply. You can also apply a manual configuration that persists only until the next reboot or driver restart (Section 3.2.5.1.2.5).

3.2.5.1.2.1 IPoIB Configuration Based on DHCP

Setting an IPoIB interface configuration based on DHCP is performed similarly to the configuration of Ethernet interfaces. In other words, you need to make sure that IPoIB configuration files include the following line:

For RedHat:

```
BOOTPROTO=dhcp
```

For IPoIB interface configuration on RHEL7 please refer to:
https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/sec-Configuring_IPoIB.html

For SLES:

```
BOOTPROTO='dchp'
```

If IPoIB configuration files are included, ifcfg-ib<n> files will be installed under:
/etc/sysconfig/network-scripts/ on a RedHat machine
/etc/sysconfig/network/ on a SuSE machine.

A patch for DHCP may be required for supporting IPoIB. For further information, please see the REAME which is available under the docs/dhcp/ directory.

Standard DHCP fields holding MAC addresses are not large enough to contain an IPoIB hardware address. To overcome this problem, DHCP over InfiniBand messages convey a client identifier field used to identify the DHCP session. This client identifier field can be used to associate an IP address with a client identifier value, such that the DHCP server will grant the same IP address to any client that conveys this client identifier.

The length of the client identifier field is not fixed in the specification. For the *Mellanox OFED for Linux* package, it is recommended to have IPoIB use the same format that FlexBoot uses for this client identifier.

3.2.5.1.2.2 DHCP Server

In order for the DHCP server to provide configuration records for clients, an appropriate configuration file needs to be created. By default, the DHCP server looks for a configuration file called `dhcpd.conf` under `/etc`. You can either edit this file or create a new one and provide its full path to the DHCP server using the -cf flag (See a file example at `docs/dhcpd.conf`).

The DHCP server must run on a machine which has loaded the IPoIB module.

To run the DHCP server from the command line, enter:

```
dhcpd <IB network interface name> -d
```

Example:

```
host1# dhcpd ib0 -d
```

3.2.5.1.2.3 DHCP Client (Optional)

A DHCP client can be used if you need to prepare a diskless machine with an IB driver.

In order to use a DHCP client identifier, you need to first create a configuration file that defines the DHCP client identifier.
Then run the DHCP client with this file using the following command:

```
dhclient -cf <client conf file> <IB network interface name>
```

Example of a configuration file for the ConnectX (PCI Device ID 26428), called `dhclient.conf`:

```
# The value indicates a hexadecimal number
interface "ib1" {
send dhcp-client-identifier
ff:00:00:00:00:00:02:00:00:02:c9:00:00:02:c9:03:00:00:10:39;
}
```

Example of a configuration file for InfiniHost III Ex (PCI Device ID 25218), called `dhclient.conf`:

```
# The value indicates a hexadecimal number
interface "ib1" {
send dhcp-client-identifier
20:00:55:04:01:fe:80:00:00:00:00:00:00:00:02:c9:02:00:23:13:92;
}
```

In order to use the configuration file, run:

```
host1# dhclient -cf dhclient.conf ib1
```

3.2.5.1.2.4 Static IPoIB Configuration

If you wish to use an IPoIB configuration that is not based on DHCP, you need to supply the installation script with a configuration file (using the '-n' option) containing the full IP configuration. The IPoIB configuration file can specify either or both of the following data for an IPoIB interface:

- A static IPoIB configuration
- An IPoIB configuration based on an Ethernet configuration

  See your Linux distribution documentation for additional information about configuring IP addresses.

The following code lines are an excerpt from a sample IPoIB configuration file:

```
# Static settings; all values provided by this file
IPADDR_ib0=11.4.3.175
NETMASK_ib0=255.255.0.0
NETWORK_ib0=11.4.0.0
BROADCAST_ib0=11.4.255.255
ONBOOT_ib0=1
# Based on eth0; each '*' will be replaced with a corresponding octet
# from eth0.
LAN_INTERFACE_ib0=eth0
IPADDR_ib0=11.4.'*'.'*'
NETMASK_ib0=255.255.0.0
NETWORK_ib0=11.4.0.0
BROADCAST_ib0=11.4.255.255
ONBOOT_ib0=1
# Based on the first eth<n> interface that is found (for n=0,1,...);
# each '*' will be replaced with a corresponding octet from eth<n>.
LAN_INTERFACE_ib0=
IPADDR_ib0=11.4.'*'.'*'
NETMASK_ib0=255.255.0.0
NETWORK_ib0=11.4.0.0
BROADCAST_ib0=11.4.255.255
ONBOOT_ib0=1
```

3.2.5.1.2.5 Manually Configuring IPoIB

This manual configuration persists only until the next reboot or driver restart.

To manually configure IPoIB for the default IB partition (VLAN), perform the following steps:

Step 1. To configure the interface, enter the `ifconfig` command with the following items:

- The appropriate IB interface (ib0, ib1, etc.)
- The IP address that you want to assign to the interface
- The netmask keyword
- The subnet mask that you want to assign to the interface

The following example shows how to configure an IB interface:

```
host1$ ifconfig ib0 11.4.3.175 netmask 255.255.0.0
```

Step 2. (Optional) Verify the configuration by entering the `ifconfig` command with the appropriate interface identifier *ib#* argument.

The following example shows how to verify the configuration:

```
host1$ ifconfig ib0
b0  Link encap:UNSPEC  HWaddr 80-00-04-04-FE-80-00-00-00-00-00-00-00-00-00-00
inet addr:11.4.3.175  Bcast:11.4.255.255  Mask:255.255.0.0
UP BROADCAST MULTICAST  MTU:65520  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:128
RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Step 3. Repeat Step 1 and Step 2 on the remaining interface(s).

3.2.5.1.3 Subinterfaces

You can create subinterfaces for a primary IPoIB interface to provide traffic isolation. Each such subinterface (also called a child interface) has a different IP and network addresses from the primary (parent) interface. The default Partition Key (PKey), ff:ff, applies to the primary (parent) interface.

This section describes how to

- Create a subinterface (Section 3.2.5.1.3.1)
- Remove a subinterface (Section 3.2.5.1.3.2)

3.2.5.1.3.1 Creating a Subinterface

In the following procedure, ib0 is used as an example of an IB subinterface.

To create a child interface (subinterface), follow this procedure:

Step 1. Decide on the PKey to be used in the subnet (valid values can be 0 or any 16-bit unsigned value). The actual PKey used is a 16-bit number with the most significant bit set. For example, a value of 1 will give a PKey with the value 0x8001.

Step 2. Create a child interface by running:

```
host1$ echo <PKey> > /sys/class/net/<IB subinterface>/create_child
```

Example:

```
host1$ echo 1 > /sys/class/net/ib0/create_child
```

This will create the interface ib0.8001.

Step 3. Verify the configuration of this interface by running:

```
host1$ ifconfig <subinterface>.<subinterface PKey>
```

Using the example of Step 2:

```
host1$ ifconfig ib0.8001
ib0.8001  Link encap:UNSPEC  HWaddr 80-00-00-4A-FE-80-00-00-00-00-00-00-00-00-00-00
BROADCAST MULTICAST  MTU:2044  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:128
RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Step 4. As can be seen, the interface does not have IP or network addresses. To configure those, you should follow the manual configuration procedure described in Section 3.2.5.1.2.5.

Step 5. To be able to use this interface, a configuration of the Subnet Manager is needed so that the PKey chosen, which defines a broadcast address, be recognized.

3.2.5.1.3.2 Removing a Subinterface

To remove a child interface (subinterface), run:

```
echo <subinterface PKey> /sys/class/net/<ib_interface>/delete_child
```

Using the example of Step 2:

```
echo 0x8001 > /sys/class/net/ib0/delete_child
```

Note that when deleting the interface you must use the PKey value with the most significant bit set (e.g., 0x8000 in the example above).

3.2.5.1.4 Verifying IPoIB Functionality

To verify your configuration and your IPoIB functionality, perform the following steps:

Step 1. Verify the IPoIB functionality by using the `ifconfig` command.

The following example shows how two IB nodes are used to verify IPoIB functionality. In the following example, IB node 1 is at 11.4.3.175, and IB node 2 is at 11.4.3.176:

```
host1# ifconfig ib0 11.4.3.175 netmask 255.255.0.0
host2# ifconfig ib0 11.4.3.176 netmask 255.255.0.0
```

Step 2. Enter the ping command from 11.4.3.175 to 11.4.3.176.

The following example shows how to enter the ping command:

```
host1# ping -c 5 11.4.3.176
PING 11.4.3.176 (11.4.3.176) 56(84) bytes of data.
```

```
64 bytes from 11.4.3.176: icmp_seq=0 ttl=64 time=0.079 ms
64 bytes from 11.4.3.176: icmp_seq=1 ttl=64 time=0.044 ms
64 bytes from 11.4.3.176: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 11.4.3.176: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 11.4.3.176: icmp_seq=4 ttl=64 time=0.065 ms
--- 11.4.3.176 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms rtt min/avg/max/mdev =
0.044/0.058/0.079/0.014 ms, pipe 2
```

3.2.5.1.5 Bonding IPoIB

To create an interface configuration script for the ibX and bondX interfaces, you should use the standard syntax (depending on your OS).

Bonding of IPoIB interfaces is accomplished in the same manner as would bonding of Ethernet interfaces: via the Linux Bonding Driver.

- Network Script files for IPoIB slaves are named after the IPoIB interfaces (e.g: ifcfg-ib0)
- The only meaningful bonding policy in IPoIB is High-Availability (bonding mode number 1, or active-backup)
- Bonding parameter "fail_over_mac" is meaningless in IPoIB interfaces, hence, the only supported value is the default: 0 (or "none" in SLES11)

For a persistent bonding IPoIB Network configuration, use the same Linux Network Scripts semantics, with the following exceptions/ additions:

- In the bonding master configuration file (e.g: ifcfg-bond0), in addition to Linux bonding semantics, use the following parameter: MTU=65520

65520 is a valid MTU value only if all IPoIB slaves operate in Connected mode (See Section 3.2.5.1.1, “IPoIB Mode Setting”, on page 140) and are configured with the same value. For IPoIB slaves that work in datagram mode, use MTU=2044. If you do not set the correct MTU or do not set MTU at all, performance of the interface might decrease.

- In the bonding slave configuration file (e.g: ifcfg-ib0), use the same Linux Network Scripts semantics. In particular: DEVICE=ib0
- In the bonding slave configuration file (e.g: ifcfg-ib0.8003), the line TYPE=InfiniBand is necessary when using bonding over devices configured with partitions (p_key)
- For RHEL users:

  In /etc/modprobe.b/bond.conf add the following lines:

  ```
  alias bond0 bonding
  ```
- For SLES users:

  It is necessary to update the MANDATORY_DEVICES environment variable in /etc/sysconfig/network/config with the names of the IPoIB slave devices (e.g. ib0, ib1, etc.). Otherwise, bonding master may be created before IPoIB slave interfaces at boot time.

  It is possible to have multiple IPoIB bonding masters and a mix of IPoIB bonding master and Ethernet bonding master. However, It is NOT possible to mix Ethernet and IPoIB slaves under the same bonding master

Restarting openibd does no keep the bonding configuration via Network Scripts. You have to restart the network service in order to bring up the bonding master. After the configuration is saved, restart the network service by running: /etc/init.d/network restart.

3.2.5.1.6 Dynamic PKey Change

Dynamic PKey change means the PKey can be changed (add/removed) in the SM database and the interface that is attached to that PKey is updated immediately without the need to restart the driver.

If the PKey is already configured in the port by the SM, the child-interface can be used immediately. If not, the interface will be ready to use only when SM adds the relevant PKey value to the port after the creation of the child interface. No additional configuration is required once the child-interface is created.

3.2.5.2 Ethernet Tunneling Over IPoIB Driver (eIPoIB)

The eth_ipoib driver provides a standard Ethernet interface to be used as a Physical Interface (PIF) into the Hypervisor virtual network, and serves one or more Virtual Interfaces (VIF). This driver supports L2 Switching (Direct Bridging) as well as other L3 Switching modes (e.g. NAT). This document explains the configuration and driver behavior when configured in Bridging mode.

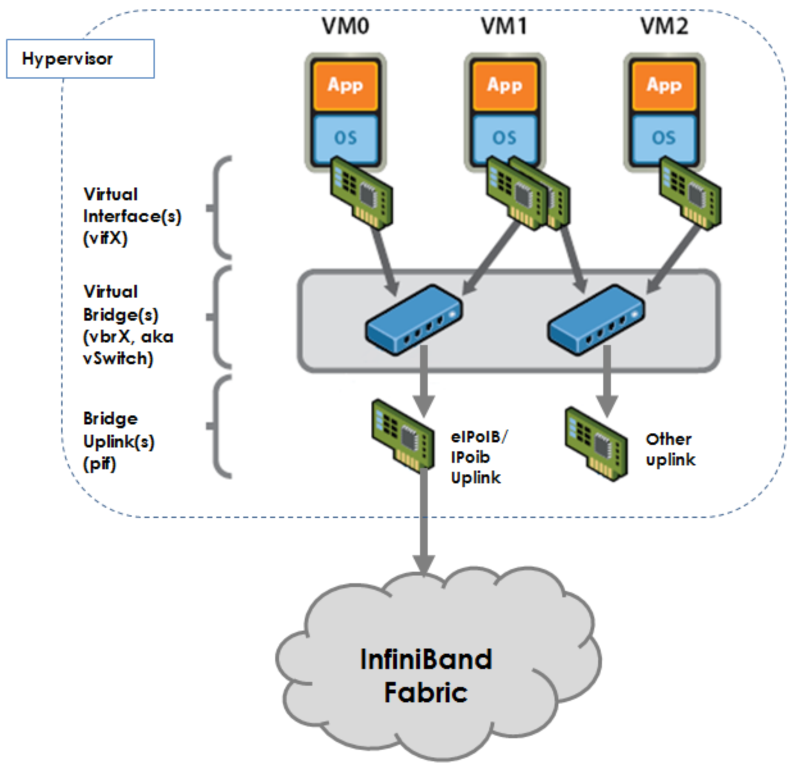
In virtualization environment, a virtual machine can be expose to the physical network by performing the next setting:

Step 1. Create a virtual bridge

Step 2. Attach the para-virtualized interface created by the eth_ipoib driver to the bridge

Step 3. Attach the Ethernet interface in the Virtual Machine to that bridge

The diagram below describes the topology that was created after these steps:



The diagram shows how the traffic from the Virtual Machine goes to the virtual-bridge in the Hypervisor and from the bridge to the eIPoIB interface. eIPoIB interface is the Ethernet interface that enslaves the IPoIB interfaces in order to send/receive packets from the Ethernet interface in the Virtual Machine to the IB fabric beneath.

3.2.5.2.1 Enabling the eIPoIB Driver

Once the mlnx_ofed driver installation is completed, perform the following:

Step 1. Open the `/etc/infiniband/openib.conf` file and include:

```
E_IPOIB_LOAD=yes
```

Step 2. Restart the InfiniBand drivers.

```
/etc/init.d/openibd restart
```

3.2.5.2.2 Configuring the Ethernet Tunneling Over IPoIB Driver

When eth_ipoib is loaded, number of eIPoIB interfaces are created, with the following default naming scheme: `ethX`, where X represents the ETH port available on the system.

To check which eIPoIB interfaces were created:

```
cat /sys/class/net/eth_ipoib_interfaces
```

For example, on a system with dual port HCA, the following two interfaces might be created; `eth4` and `eth5`.

```
cat /sys/class/net/eth_ipoib_interfaces
eth4 over IB port: ib0
eth5 over IB port: ib1
```

These interfaces can be used to configure the network for the guest. For example, if the guest has a VIF that is connected to the Virtual Bridge `br0`, then enslave the eIPoIB interface to `br0` by running:

```
# brctl addif br0 ethX
```

In RHEL KVM environment, there are other methods to create/configure your virtual network (e.g. macvtap). For additional information, please refer to the Red Hat User Manual.

The IPoIB daemon (ipoibd) detects the new virtual interface that is attached to the same bridge as the eIPoIB interface and creates a new IPoIB instances for it in order to send/receive data. As a result, number of IPoIB interfaces (ibX.Y) are shown as being created/destroyed, and are being enslaved to the corresponding `ethX` interface to serve any active VIF in the system according to the set configuration, This process is done automatically by the `ipoibd` service.

➢ ***To see the list of IPoIB interfaces enslaved under eth_ipoib interface.***

```
# cat /sys/class/net/ethX/eth/vifs
```

For example:

```
# cat /sys/class/net/eth5/eth/vifs
  SLAVE=ib0.1      MAC=9a:c2:1f:d7:3b:63 VLAN=N/A
  SLAVE=ib0.2      MAC=52:54:00:60:55:88 VLAN=N/A
  SLAVE=ib0.3      MAC=52:54:00:60:55:89 VLAN=N/A
```

Each ethX interface has at lease one ibX.Y slave to serve the PIF itself. In the VIFs list of ethX you will notice that ibX.1 is always created to serve applications running from the Hypervisor on top of the ethX interface directly.

For InfiniBand applications that require native IPoIB interfaces (e.g. CMA), the original IPoIB interfaces ibX can still be used. For example, CMA and ethX drivers can co-exist and make use of IPoIB ports; CMA can use ib0, while eth0.ipoib interface will use ibX.Y interfaces.

➢ ***To see the list of eIPoIB interfaces.***

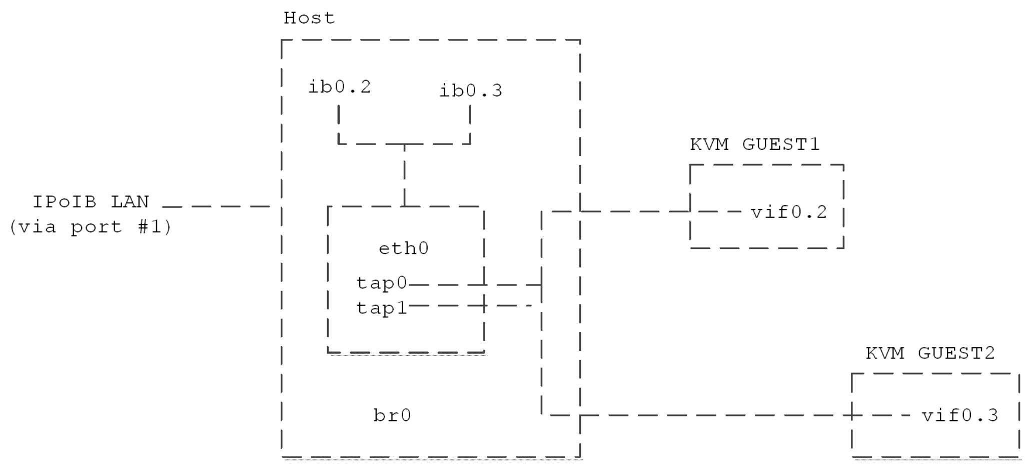
```
# cat /sys/class/net/eth_ipoib_interfaces
```

For example:

```
# cat /sys/class/net/eth_ipoib_interfaces
  eth4 over IB port: ib0
  eth5 over IB port: ib1
```

The example above shows, two eIPoIB interfaces, where eth4 runs traffic over ib0, and eth5 runs traffic over ib1.

Figure 6: An Example of a Virtual Network



The example above shows a few IPoIB instances that serve the virtual interfaces at the Virtual Machines.

To display the services provided to the Virtual Machine interfaces:

```
# cat /sys/class/net/eth0/eth/vifs
```

Example:

```
# cat /sys/class/net/eth0/eth/vifs
  SLAVE=ib0.2 MAC=52:54:00:60:55:88 VLAN=N/A
```

In the example above the ib0.2 IPoIB interface serves the MAC 52:54:00:60:55:88 with no VLAN tag for that interface.

3.2.5.2.3 VLAN Configuration Over an eIPoIB Interface

eIPoIB driver supports VLAN Switch Tagging (VST) mode, which enables the virtual machine interface to have no VLAN tag over it, thus allowing VLAN tagging to be handled by the Hypervisor.

➢ ***To attach a Virtual Machine interface to a specific isolated tag:***

Step 1. Verify the VLAN tag to be used has the same pkey value that is already configured on that ib port.

```
# cat /sys/class/infiniband/mlx4_0/ports/<ib port>/pkeys/*
```

Step 2. Create a VLAN interface in the Hypervisor, over the eIPoIB interface.

```
# vconfig add <eIPoIB interface> <vlan tag>
```

Step 3. Attach the new VLAN interface to the same bridge that the virtual machine interface is already attached to.

```
# brctl addif <br-name> <interface-name>
```

For example, to create the VLAN tag 3 with pkey 0x8003 over that port in the eIPoIB interface eth4, run:

```
#vconfig add eth4 3
#brctl addif br2 eth4.3
```

3.2.5.2.4 Setting Performance Tuning

- Use 4K MTU over OpenSM.

```
Default=0xffff, ipoib, mtu=5 : ALL=full;
```

- Use MTU for 4K (4092 Bytes):
  - In UD mode, the maximum MTU value is 4092 Bytes

  Make sure that all interfaces (including the guest interface and its virtual bridge) have the same MTU value (MTU 4092 Bytes). For further information of MTU settings, please refer to the Hypervisor User Manual.
- Tune the TCP/IP stack using sysctl (dom0/domu)

```
# /sbin/sysctl_perf_tuning
```

- Other performance tuning for KVM environment such as vCPU pinning and NUMA tuning may apply. For further information, please refer to the Hypervisor User Manual.

3.2.5.3 Ethernet over IB (EoIB)

Ethernet over IB (EoIB) is currently supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only.

The Ethernet over IB (EoIB) mlx4_vnic module is a network interface implementation over InfiniBand. EoIB encapsulates Layer 2 datagrams over an InfiniBand Datagram (UD) transport service. The InfiniBand UD datagrams encapsulates the entire Ethernet L2 datagram and its payload.

To perform this operation the module performs an address translation from Ethernet layer 2 MAC addresses (48 bits long) to InfiniBand layer 2 addresses made of LID/GID and QPN. This translation is totally invisible to the OS and user. Thus, differentiating EoIB from IPoIB which exposes a 20 Bytes HW address to the OS. The mlx4_vnic module is designed for Mellanox's ConnectX® family of HCAs and intended to be used with Mellanox's BridgeX® gateway family. Having a BridgeX gateway is a requirement for using EoIB. It performs the following operations:

- Enables the layer 2 address translation required by the mlx4_vnic module.
- Enables routing of packets from the InfiniBand fabric to a 1 or 10 GigE Ethernet subnet.

3.2.5.3.1 Ethernet over IB Topology

EoIB is designed to work over an InfiniBand fabric and requires the presence of two entities:

- Subnet Manager (SM)

  The required subnet manager configuration is not unique to EoIB but rather similar to other InfiniBand applications and ULPs.
- BridgeX gateway

  The BridgeX gateway is at the heart of EoIB. On one side, usually referred to as the "internal" side, it is connected to the InfiniBand fabric by one or more links. On the other side, usually referred to as the "external" side, it is connected to the Ethernet subnet by one or more ports. The Ethernet connections on the BridgeX's external side are called external ports or eports. Every BridgeX that is in use with EoIB needs to have one or more eports connected.

3.2.5.3.1.1 External Ports (eports) and Gateway

The combination of a specific BridgeX box and a specific eport is referred to as a gateway. The gateway is an entity that is visible to the EoIB host driver and is used in the configuration of the network interfaces on the host side. For example, in the host administered vNics the user will request to open an interface on a specific gateway identifying it by the BridgeX box and eport name.

Distinguishing between gateways is essential because they determine the network topology and affect the path that a packet traverses between hosts. A packet that is sent from the host on a specific EoIB interface will be routed to the Ethernet subnet through a specific external port connection on the BridgeX box.

3.2.5.3.1.2 Virtual Hubs (vHubs)

Virtual hubs connect zero or more EoIB interfaces (on internal hosts) and an eport through a virtual hub. Each vHub has a unique virtual LAN (VLAN) ID. Virtual hub participants can send packets to one another directly without the assistance of the Ethernet subnet (external side) routing. This means that two EoIB interfaces on the same vHub will communicate solely using the InfiniBand fabric. EoIB interfaces residing on two different vHubs (whether on the same gateway or not) cannot communicate directly.

There are two types of vHubs:

- a default vHub (one per gateway) without a VLAN ID
- vHubs with unique different VLAN IDs

Each vHub belongs to a specific gateway (BridgeX® + eport), and each gateway has one default vHub, and zero or more VLAN-associated vHubs. A specific gateway can have multiple vHubs distinguishable by their unique VLAN ID. Traffic coming from the Ethernet side on a specific eport will be routed to the relevant vHub group based on its VLAN tag (or to the default vHub for that GW if no vLan ID is present).

3.2.5.3.1.3 Virtual NIC (vNic)

A virtual NIC is a network interface instance on the host side which belongs to a single vHub on a specific GW. The vNic behaves similar to any regular hardware network interface. The host can have multiple interfaces that belong to the same vHub.

3.2.5.3.2 EoIB Configuration

mlx4_vnic module supports two different modes of configuration which is passed to the host mlx4_vnic driver using the EoIB protocol:

- host administration where the vNic is configured on the host side
- network administration where the configuration is done by the BridgeX

Both modes of operation require the presence of a BridgeX gateway in order to work properly. The EoIB driver supports a mixture of host and network administered vNics.

3.2.5.3.2.1 EoIB Host Administered vNic

In the host administered mode, vNics are configured using static configuration files located on the host side. These configuration files define the number of vNics, and the vHub that each host administered vNic will belong to (i.e., the vNic's BridgeX box, eport and VLAN id properties). The mlx4_vnic_confd service is used to read these configuration files and pass the relevant data to the mlx4_vnic module. EoIB Host Administered vNic supports two forms of configuration files:

- “Central Configuration File - /etc/infiniband/mlx4_vnic.conf”
- “vNic Specific Configuration Files - ifcfg-ethX”

Both forms of configuration supply the same functionality. If both forms of configuration files exist, the central configuration file has precedence and only this file will be used.

3.2.5.3.2.2 Central Configuration File - /etc/infiniband/mlx4_vnic.conf

The mlx4_vnic.conf file consists of lines, each describing one vNic. The following file format is used:

```
name=eth47 mac=00:25:8B:27:16:84 ib_port=mlx4_0:1 vid=2 vnic_id=7 bx=BX001 eport=A11
```

The fields used in the file have the following meaning:

Table 3 - mlx4_vnic.conf file format

Field	Description
name	The name of the interface that is displayed when running ifconfig.
mac	The mac address to assign to the vNic.
ib_port	The device name and port number in the form [device name]:[port number]. The device name can be retrieved by running ibv_devinfo and using the output of hca_id field. The port number can have a value of 1 or 2.
vid	[Optional field] If VLAN ID exists, the vNic will be assigned the specified VLAN ID. This value must be between 0 and 4095. • If the vid is set to 'all', the ALL-VLAN mode will be enabled and the vNic will support multiple vNic tags. • If no vid is specified or value -1 is set, the vNic will be assigned to the default vHub associated with the GW.
vnic_id	A unique number per vNic between 0 and 16K.
bx	The BridgeX box system GUID or system name string.
eport	The string describing the eport name.
pkey	[Optional field] If discovery_pkey module parameter is set, this value will control which partitions would be used to discover the gateways. For more information about discovery_pkeys please refer to Section 3.2.5.3.3.5, “Discovery Partitions Configuration”, on page 158

3.2.5.3.2.3 vNic Specific Configuration Files - ifcfg-ethX

EoIB configuration can use the ifcfg-ethX files used by the network service to derive the needed configuration. In such case, a separate file is required per vNic. Additionally, you need to update the ifcfg-ethX file and add some new attributes to it.

On Red Hat Linux, the new file will be of the form:

```
DEVICE=eth2
HWADDR=00:30:48:7d:de:e4
BOOTPROTO=dhcp
```

```
ONBOOT=yes
BXADDR=BX001
BXEPORT=A10
VNICIBPORT=mlx4_0:1
VNICVLAN=3  (Optional field)
GW_PKEY=0xfff1
```

The fields used in the file for vNic configuration have the following meaning:

Table 4 - Red Hat Linux mlx4_vnic.conf file format

Field	Description
DEVICE	An optional field. The name of the interface that is displayed when running ifconfig. If it is not present, the trailer of the configuration file name (e.g. ifcfg-eth47 => "eth47") is used instead.
HWADDR	The mac address to assign the vNic.
BXADDR	The BridgeX box system GUID or system name string.
BXEPORT	The string describing the eport name.
VNICVLAN	[Optional field] If it exists, the vNic will be assigned the VLAN ID specified. This value must be between 0 and 4095 or 'all' for ALL-VLAN feature.
VNICIBPORT	The device name and port number in the form [device name]:[port number]. The device name can be retrieved by running ibv_devinfo and using the output of hca_id field. The port number can have a value of 1 or 2.
GW_PKEY	[Optional field] If discovery_pkey module parameter is set, this value will control on what partition would be used to discover the gateways. For more information about discovery_pkeys please refer to Section 3.2.5.3.3.5, "Discovery Partitions Configuration", on page 158

Other fields available for regular eth interfaces in the ifcfg-ethX files may also be used.

3.2.5.3.2.4 mlx4_vnic_confd

Once the configuration files are updated, the host administered vNics can be created. To manage the host administrated vNics, run the following script:

```
Usage: /etc/init.d/mlx4_vnic_confd {start|stop|restart|reload|status}
```

This script manages host administrated vNics only, to retrieve general information about the vNics on the system including network administrated vNics, refer to Section 3.2.5.3.3.1, "mlx-4_vnic_info", on page 157.

When using BXADDR/bx field, all vNics BX address configuration should be consistent: either all of them use GUID format, or name format.

The MAC and VLAN values are set using the configuration files only, other tools such as (vconfig) for VLAN modification, or (ifconfig) for MAC modification, are not supported.

3.2.5.3.2.5 EoIB Network Administered vNic

In network administered mode, the configuration of the vNic is done by the BridgeX®. If a vNic is configured for a specific host, it will appear on that host once a connection is established between the BridgeX and the mlx4_vnic module. This connection between the mlx4_vnic modules and all available BridgeX boxes is established automatically when the mlx4_vnic module is loaded. If the BridgeX is configured to remove the vNic, or if the connection between the host and BridgeX is lost, the vNic interface will disappear (running ifconfig will not display the interface). Similar to host administered vNics, a network administered, vNic resides on a specific vHub.

For further information on how to configure a network administered vNic, please refer to BridgeX documentation.

To disable network administered vNics on the host side load mlx4_vnic module with the net_admin module parameter set to 0.

3.2.5.3.2.6 VLAN Configuration

A vNic instance is associated with a specific vHub group. This vHub group is connected to a BridgeX external port and has a VLAN tag attribute. When creating/configuring a vNic you define the VLAN tag it will use via the vid or the VNICVLAN fields (if these fields are absent, the vNic will not have a VLAN tag). The vNic's VLAN tag will be present in all EoIB packets sent by the vNics and will be verified on all packets received on the vNic. When passed from the InfiniBand to Ethernet, the EoIB encapsulation will be disassembled but the VLAN tag will remain.

For example, if the vNic "eth23" is associated with a vHub that uses BridgeX "bridge01", eport "A10" and VLAN tag 8, all incoming and outgoing traffic on eth23 will use a VLAN tag of 8. This will be enforced by both BridgeX and destination hosts. When a packet is passed from the internal fabric to the Ethernet subnet through the BridgeX it will have a "true" Ethernet VLAN tag of 8.

The VLAN implementation used by EoIB uses operating systems unaware of VLANs. This is in many ways similar to switch tagging in which an external Ethernet switch adds/strips tags on traffic preventing the need of OS intervention. EoIB does not support OS aware VLANs in the form of vconfig.

3.2.5.3.2.7 Configuring VLANs

To configure VLAN tag for a vNic, add the VLAN tag property to the configuration file in host administrated mode, or configure the vNic on the appropriate vHub in network administered mode. In the host administered mode when a vHub with the requested VLAN tag is not available, the vNIC's login request will be rejected.

- Host administered VLAN configuration in centralized configuration file can be modified as follow:

  Add "vid=<VLAN tag>" or remove vid property for no VLAN

- Host administered VLAN configuration with ifcfg-ethX configuration files can be modified as follow:

Add "VNICVLAN=<VLAN tag>" or remove VNICVLAN property for no VLAN

Using a VLAN tag value of 0 is not recommended because the traffic using it would not be separated from non VLAN traffic.

For Host administered vNics, VLAN entry must be set in the BridgeX first. For further information, please refer to BridgeX® documentation.

3.2.5.3.2.8 EoIB Multicast Configuration

Configuring Multicast for EoIB interfaces is identical to multicast configuration for native Ethernet interfaces.

EoIB maps Ethernet multicast addresses to InfiniBand MGIDs (Multicast GID). It ensures that different vHubs use mutually exclusive MGIDs. Thus preventing vNics on different vHubs from communicating with one another.

3.2.5.3.2.9 EoIB and Quality of Service

EoIB enables the use of InfiniBand service levels. The configuration of the SL is performed through the BridgeX and lets you set different data/control service level values per BridgeX® box.

For further information on the use of non default service levels, please refer to BridgeX documentation.

3.2.5.3.2.10 IP Configuration Based on DHCP

Setting an EoIB interface configuration based on DHCP (v3.1.2 which is available via www.isc.org) is performed similarly to the configuration of Ethernet interfaces. When setting the EoIB configuration files, verify that it includes following lines:

- For RedHat: BOOTPROTO=dhcp
- For SLES: BOOTPROTO='dchp'

If EoIB configuration files are included, ifcfg-eth<n> files will be installed under /etc/sysconfig/network-scripts/ on a RedHat machine and under /etc/sysconfig/network/ on a SuSE machine.

3.2.5.3.2.11 DHCP Server

Using a DHCP server with EoIB does not require special configuration. The DHCP server can run on a server located on the Ethernet side (using any Ethernet hardware) or on a server located on the InfiniBand side and running EoIB module.

3.2.5.3.2.12 Static EoIB Configuration

To configure a static EoIB you can use an EoIB configuration that is not based on DHCP. Static configuration is similar to a typical Ethernet device configuration. For further information on how to configure IP addresses, please refer to your Linux distribution documentation.

Ethernet configuration files are located at /etc/sysconfig/network-scripts/ on a RedHat machine and at /etc/sysconfig/network/ on a SuSE machine.

3.2.5.3.2.13 Sub Interfaces (VLAN)

EoIB interfaces do not support creating sub interfaces via the vconfig command, unless working in ALL VLAN mode.. To create interfaces with VLAN, refer to Section 3.2.5.3.2.7, “Configuring VLANs”, on page 155.

3.2.5.3.3 Retrieving EoIB Information

3.2.5.3.3.1 mlx4_vnic_info

To retrieve information regarding EoIB interfaces, use the script mlx4_vnic_info. This script provides detailed information about a specific vNic or all EoIB vNic interfaces, such as: BX info, IOA info, SL, PKEY, Link state and interface features. If network administered vNics are enabled, this script can also be used to discover the available BridgeX® boxes from the host side.

- To discover the available gateway, run:

```
mlx4_vnic_info -g
```

- To receive the full vNic information of eth10, run:

```
mlx4_vnic_info -i eth10
```

- To receive a shorter information report on eth10, run:

```
mlx4_vnic_info -s eth10
```

- To get help and usage information, run:

```
mlx4_vnic_info --help
```

3.2.5.3.3.2 Link State

An EoIB interface can report two different link states:

- The physical link state of the interface that is made up of the actual HCA port link state and the status of the vNics connection with the BridgeX®. If the HCA port link state is down or the EoIB connection with the BridgeX has failed, the link will be reported as down because without the connection to the BridgeX the EoIB protocol cannot work and no data can be sent on the wire. The mlx4_vnic driver can also report the status of the external BridgeX port status by using the mlx4_vnic_info script. If the eport_state_enforce module parameter is set, then the external port state will be reported as the vNic interface link state. If the connection between the vNic and the BridgeX is broken (hence the external port state is unknown)the link will be reported as down.
- the link state of the external port associated with the vNic interface

A link state is down on a host administrated vNic, when the BridgeX is connected and the InfiniBand fabric appears to be functional. The issue might result from a misconfiguration of either BXADDR or/and BXEPORT configuration file.

To query the link state run the following command and look for "Link detected":

```
ethtool <interface name>
```

Example:

```
ethtool eth10
Settings for eth10:
Supported ports: [ ]
Supported link modes:
Supports auto-negotiation: No
Advertised link modes:  Not reported
Advertised auto-negotiation: No
Speed: Unknown! (10000)
Duplex: Full
Port: Twisted Pair
PHYAD: 0
Transceiver: internal
Auto-negotiation: off
Supports Wake-on: d
Wake-on: d
Current message level: 0x00000000 (0)
Link detected: yes
```

3.2.5.3.3.3 Bonding Driver

EoIB uses the standard Linux bonding driver. For more information on the Linux Bonding driver please refer to:

<kernel-source>/Documentation/networking/bonding.txt.

Currently only fail-over modes are supported by the EoIB driver, load-balancing modes including static and dynamic (LACP) configurations are not supported.

3.2.5.3.3.4 Jumbo Frames

EoIB supports jumbo frames up to the InfiniBand limit of 4K bytes. The default Maximum Transmit Unit (MTU) for EoIB driver is 1500 bytes.

➢ ***To configure EoIB to work with jumbo frames:***

1. Make sure that the IB HCA and Switches hardware support 4K MTU.
2. Configure Mellanox low level driver to support 4K MTU. Add:

```
mlx4_core module parameter to set_4k_mtu=1
```

3. Change the MTU value of the vNic, for example, run:

```
 ifconfig eth2 mtu 4038
```

Due to EoIB protocol overhead, the maximum MTU value that can be set for the vNic interface is: 4038 bytes. If the vNic is configured to use VLANs, then the maximum MTU is: 4034 bytes (due to VLAN header insertion).

3.2.5.3.3.5 Discovery Partitions Configuration

EoIB enables mapping of VLANs to InfiniBand partitions. Mapping VLANs to partitions causes all EoIB data traffic and all vNic related control traffic to be sent to the mapped partitions. In rare cases, it might be useful to ensure that EoIB discovery packets (packets used for discovery of

Gateways (GWs) and vice versa) are sent to a non default partition. This might be used to limit and enforce the visibility of GWs by different hosts.

The **`discovery_pkeys`** module parameter can be used to define which partitions would be used to discove the GWs. The module parameters allow the using of up to 24 different PKEYs. If not set, the default PKEY will be used, and only GWs using the default PKEY would be discovered.

For example, to configure a host to discover GWs on three partitions 0xffff,0xfff1 and 0x3 add the following line to modprobe configuration file:

```
options mlx4_vnic discovery_pkeys=0xffff,0xfff1,0x3
```

When using this feature combined with host administrated vnics, each vnic should also be configured with the partition it should be created on.

For example, for creating host admin vnic on I/F eth20, with pkey 0xfff1 add the following line to ifcg-eth20:

```
GW_PKEY=0xfff1
```

When using a non default partition, the GW partitions should also be configured on the GW in the BridgeX. Additionally, the Subnet Manager must be configured accordingly.

3.2.5.3.3.6 ALL VLAN

In Ethernet over InfiniBand (EoIB), a vNic is a member of a vHUB that uniquely defines its Virtual Local Area Networks (VLAN) tag. The VLAN tag is used in the VLAN header within the EoIB packets, and is enforced by EoIB hosts when handling the EoIB packets. The tag is also extended to the Ethernet fabric when packets pass through the BridgeX®. This model of operation ensures a high level of security however, it requires each VLAN tag used to have its own individual vNic to be created and each vHub requires InfiniBand fabric resources like multicast groups (MGIDs).

If many VLANs are needed, the resources required to create and manage them are large. ALL VLAN vHub enables the user to use its resources efficiently by creating a vNic that can support multiple VLAN tags without creating multiple vNics. However, it reduces VLAN separation compared to the vNic /vHub model.

3.2.5.3.3.7 ALL VLAN Functionality

When ALL VLAN is enabled, the address lookup on the BridgeX® consists of the MAC address only (without the VLAN), so all packets with the same MAC regardless of the VLAN, are sent to the same InfiniBand address. Same behavior can be expected from the host EoIB driver, which also sends packets to the relevant InfiniBand addresses while disregarding the VLAN. In both scenarious, the Ethernet packet that is embedded in the EoIB packet includes the VLAN header enabling VLAN enforcement either in the Ethernet fabric or at the receiving EoIB host.

ALL VLAN must be supported by both the BridgeX® and by the host side.

When enabling ALL VLAN, all gateways (LAG or legacy) that have eports belonging to a gateway group (GWG) must be configured to the same behavior.
For example it is impossible to have gateway A2 configured to all-vlan mode and A3 to regular mode, because both belong to GWG A.

A gateway that is configured to work in ALL VLAN mode cannot accept login requests from
- vNics that do not support this mode
- host admin vNics that were not configured to work in ALL VLAN mode, by setting the vlan-id value to a 'all' as as described in Section 3.2.5.3.3.8, "Creating vNICs that Support ALL VLAN Mode", on page 160.

3.2.5.3.3.8 Creating vNICs that Support ALL VLAN Mode

VLANs are created on a vNIC that supports ALL VLAN mode using "vconfig".

- **net-admin vNics**

  The net-admin vNic supports ALL VLAN mode once it is created on a gateway configured with All-VLAN mode.

- **host-admin vNics**
  - To create an ALL VLAN vnic, set the VLAN's ID to 'all'.

    A gateway that is configured to work in ALL VLAN mode, can only accept login requests from hosts that are also working in a VLAN mode. e.g. the VLAN ID must be set to 'all'.

  This is an example of how to create an ALL VLAN vNic using the mlx4_vnic.conf file:

```
  name=eth44 mac=00:25:8B:27:14:78 ib_port=mlx4_0:1 vid=all vnic_id=5
bx=00:00:00:00:00:00:04:B2 eport=A10
```

  - To create an All-VLAN vNic using a specific configuration file, add the following line to the configuration file:

```
VNICVLAN=all
```

For further information on how to create host-admin vNics, please see Section 3.2.5.3.2.1, "EoIB Host Administered vNic", on page 152 EoIB Host Administered vNic.

3.2.5.3.3.9 Checking the Configuration

To verify the gateway / vNic is configured with the All-VLAN mode, use the mlx4_vnic_info script.

- **Gateway Support**

  To verify the gateway is configured to All-VLAN mode. Run:

```
mlx4_vnic_info -g <GW-NAME>
```

Example:

```
# mlx4_vnic_info -g A2
IOA_PORT      mlx4_0:1
BX_NAME       bridge-119c64
BX_GUID       00:02:c9:03:00:11:61:67
EPORT_NAME    A2
EPORT_ID      63
STATE         connected
GW_TYPE       LEGACY
PKEY          0xffff
ALL_VLAN      yes
```

- **vNic Support**

  To verify the vNIC is configured to All-VLAN mode. Run:

  ```
  mlx4_vnic_info -i <interface>
  ```

  Example:

  ```
  # mlx4_vnic_info -i eth204
  NETDEV_NAME   eth204
  NETDEV_LINK   up
  NETDEV_OPEN   yes
  GW_TYPE       LEGACY
  ALL_VLAN      yes
  ```

For further information on mlx4_vnic_info script, please see Section 3.2.5.3.3.1, “mlx-4_vnic_info”, on page 157.

3.2.5.3.4 Advanced EoIB Settings

3.2.5.3.4.1 Module Parameters

The mlx4_vnic driver supports the following module parameters. These parameters are intended to enable more specific configuration of the mlx4_vnic driver to customer needs. The mlx4_vnic is also effected by module parameters of other modules such as set_4k_mtu of mlx4_core. This modules are not addressed in this section.

The available module parameters include:

- tx_rings_num: Number of TX rings, use 0 for #cpus [default 0, max 32]
- tx_rings_len: Length of TX rings, must be power of two [default 1024, max 8K]
- rx_rings_num: Number of RX rings, use 0 for #cpus [default 0, max 32]
- rx_rings_len: Length of RX rings, must be power of two [default 2048, max 8K]
- vnic_net_admin: Network administration enabled [default 1]
- lro_num: Number of LRO sessions per ring, use 0 to disable LRO [default 32, max 32]
- eport_state_enforce: Bring vNic up only when corresponding External Port is up [default 0]
- discovery_pkeys: Vector of up to 24 PKEYs to be used for discovery [default 0xFFFF] (array of int)

For all module parameters list and description, run:

```
mlx4_vnic_info -I
```

To check the current module parameters, run:

```
mlx4_vnic_info -P
```

Default RX/TX rings number is the number of logical CPUs (threads). To set non-default values to module parameters, the following line should be added to modprobe configuration file (e.g. /etc/modprobe.conf file):

```
options mlx4_vnic <param_name>=<value> <param_name>=<value> ...
```

For additional information about **`discovery_pkeys`** please refer to Section 3.2.5.3.3.5, “Discovery Partitions Configuration”, on page 158

3.2.5.3.4.2 vNic Interface Naming

The mlx4_vnic driver enables the kernel to determine the name of the registered vNic. By default, the Linux kernel assigns each vNic interface the name eth<N>, where <N> is an incremental number that keeps the interface name unique in the system. The vNic interface name may not remain consistent among hosts or BridgeX reboots as the vNic creation can happen in a different order each time. Therefore, the interface name may change because of a "first-come-first-served" kernel policy. In automatic network administered mode, the vNic MAC address may also change, which makes it difficult to keep the interface configuration persistent.

To control the interface name, you can use standard Linux utilities such as IFRENAME(8), IP(8) or UDEV(7). For example, to change the interface eth2 name to eth.bx01.a10, run:

```
ifrename -i eth2 -n eth.bx01.a10
```

To generate a unique vNic interface name, use the mlx4_vnic_info script with the '-u' flag. The script will generate a new name based on the scheme:

```
eth<pci-id>.<ib-port-num>.<gw_port_id>.[vlan-id]
```

For example, if vNic eth2 resides on an InfiniBand card on the PCI BUS ID 0a:00.0 PORT #1, and is connected to the GW PORT ID #3 without VLAN, its unique name will be:

```
mlx4_vnic_info -u eth2
eth2   eth10.1.3
```

You can add your own custom udev rule to use the output of the script and to rename the vNic interfaces automatically. To create a new udev rule file under /etc/udev/rules.d/61-vnic-net.rules, include the line:

```
SUBSYSTEM=="net", PROGRAM=="/sbin/mlx4_vnic_info -u %k", NAME="%c{2+}"
```

UDEV service is active by default however if it is not active, run:

```
/sbin/udevd -d
```

When vNic MAC address is consistent, you can statically name each interface using the UDEV following rule:

```
SUBSYSTEM=="net", SYSFS{address}=="aa:bb:cc:dd:ee:ff", NAME="ethX"
```

For further information on the UDEV rules syntax, please refer to udev man pages.

3.2.5.3.4.3 Para-Virtualized vNic

EoIB driver interfaces can be also used for Linux based virtualization environment such as Xen/KVM based Hypervisors. This section explains how to configure Para-Virtualized (PV) EoIB to work in such an environment.

3.2.5.3.4.4 Driver Configuration

For PV-EoIB to work properly, the following features must be disabled in the driver:

- Large Receive Offload (LRO)
- TX completion polling
- RX fragmented buffers

To disable the features above, edit the modprobe configuration file as follow:

```
options mlx4_vnic lro_num=0 tx_polling=0 rx_linear=1
```

For the full list of mlx4_vnic module parameters, run:

```
# modinfo mlx4_vnic
```

3.2.5.3.4.5 Network Configuration

PV-EoIB supports both L2 (bridged) and L3 (routed) network models. The 'physical' interfaces that can be enslaved to the Hypervisor virtual bridge are actually EoIB vNics, and they can be created as on an native Linux machine. PV-EoIB driver supports both host-administrated and network-administrated vNics. Please refer to Section 3.2.5.3.2, “EoIB Configuration”, on page 152 for more information on vNics configuration.

Once an EoIB vNic is enslaved to a virtual bridge, it can be used by any Guest OS that is supported by the Hypervisor. The driver will automatically manage the resources required to serve the Guest OS network virtual interfaces (based on their MAC address).

To see the list of MAC addresses served by an EoIB vNic, log into the Hypervisor and run the command:

```
# mlx4_vnic_info -m <interface>
```

The driver detects virtual interfaces MAC addresses based in their outgoing packets, so you may notice that the virtual MAC address is being detected by the EoIB driver only after the first packet is sent out by the Guest OS. Virtual resources MAC addresses cleanup is managed by mlx4_vnic daemon as explained in Section 3.2.5.3.4.9, “Resources Cleanup”, on page 164.

3.2.5.3.4.6 Multicast Configuration

Virtual machines multicast traffic over PV-EoIB is supported in promiscuous mode. Hence, all multicast traffic is sent over the broadcast domain, and filtered in the VM level.

- To enable promiscuous multicast, log into the BridgeX CLI and run the command:

  ```
  # bxm eoib mcast promiscuous
  ```

  Please refer to BridgeX CLI Guide for additional details.
- To see the multicast configuration of a vNic from the host, log into the Hypervisor and run:

  ```
  # mlx4_vnic_info -i <interface> | grep MCAST
  ```

3.2.5.3.4.7 VLANs

Virtual LANs are supported in EoIB vNic level, where VLAN tagging/untagging is done by the EoIB driver.

- To enable VLANs on top of a EoIB vNic:
  a. Create a new vNic interface with the corresponding VLAN ID

b. Enslave it to a virtual bridge to be used by the Guest OS. The VLAN tagging/untagging is transparent to the Guest and managed in EoIB driver level.

The vconfig utility is not supported by EoIB driver, a new vNic instance must be created instead. For further information, see Section 3.2.5.3.2.6, “VLAN Configuration”, on page 155.

Virtual Guest Tagging (VGT) is not supported. The model explained above applies to Virtual Switch Tagging (VST) only.

3.2.5.3.4.8 Migration

Some Hypervisors provide the ability to migrate a virtual machine from one physical server to another, this feature is seamlessly supported by PV-EoIB. Any network connectivity over EoIB will automatically be resumed on the new physical server. The downtime that may occur during this process is minor.

3.2.5.3.4.9 Resources Cleanup

When a virtual interface within the Guest OS is no longer connected to an EoIB link, its MAC address need to be removed from the EoIB driver. The cleaning is managed by the Garbage Collector (GC) service. The GC functionality is included in the mlx4_vnic daemon (python script):

```
# /sbin/mlx4_vnicd
```

- To enable/disable the mlx4_vnic daemon,
  a. Edit the **`/etc/infiniband/mlx4_vnic.conf`** file by including the line:
  ```
  # mlx4_vnicd=<yes|no> [parameters]
  ```
  b. Start the service **`mlx4_vnic_confd`** to read and apply the configuration:
  ```
  # /etc/init.d/mlx4_vnic_confd start
  ```
- To see full list of the daemon parameters, run:
  ```
  # mlx4_vnicd --help
  ```

For example, to enable mlx4_vnic daemon with GC:

```
# cat /etc/infiniband/mlx4_vnic.conf
  mlx4_vnicd=yes gc_enable=yes
# /etc/init.d/mlx4_vnic_confd start
  Checking configuration file:                          [  OK  ]
  Starting mlx4_vnicd (pid 30920):                      [  OK  ]
```

The mlx4_vnicd daemon requires xenstore or libvirt to run.

Some Hypervisors may not have enough memory for the driver domain, as a result **`mlx4_vnic`** driver may fail to initialize or create more vNics, causing the machine to be unresponsive.

- To avoid this behavior, you can:
  a. Allocate more memory for the driver domain.

For further information on how to increase dom0_mem, please refer to:

http://support.citrix.com/article/CTX126531

b. Lower the **`mlx4_vnic`** driver memory consumption by decreasing its RX/TX rings number and length,

For further information, please refer to Section 3.2.5.3.4.1, “Module Parameters”, on page 161.

3.2.6 Advanced Transport

3.2.6.1 Atomic Operations

3.2.6.1.1 Atomic Operations in mlx5 Driver

Atomic Operations in Connect-IB® (mlx5 driver) are fully supported on big-endian machines (e.g. PPC). Their support is limited on little-endian machines (e.g. x86)

When using `ibv_exp_query_device` on little-endian machines with Connect-IB® the `attr.exp_atomic_cap` is set to I`BV_EXP_ATOMIC_HCA_REPLY_BE` which indicates that if enabled, the atomic operation replied value is big-endian and contradicts the host endianness.

To enable atomic operation with this endianness contradiction use the `ibv_exp_create_qp` to create the QP and set the `IBV_EXP_QP_CREATE_ATOMIC_BE_REPLY` flag on `exp_create_flags`.

3.2.6.1.2 Enhanced Atomic Operations

ConnectX® implements a set of Extended Atomic Operations beyond those defined by the IB spec. Atomicity guarantees, Atomic Ack generation, ordering rules and error behavior for this set of extended Atomic operations is the same as that for IB standard Atomic operations (as defined in section 9.4.5 of the IB spec).

3.2.6.1.2.1 Masked Compare and Swap (MskCmpSwap)

The MskCmpSwap atomic operation is an extension to the CmpSwap operation defined in the IB spec. MskCmpSwap allows the user to select a portion of the 64 bit target data for the "compare" check as well as to restrict the swap to a (possibly different) portion. The pseudocode below describes the operation:

```
| atomic_response = *va
| if (!((compare_add ^ *va) & compare_add_mask)) then
|     *va = (*va & ~(swap_mask)) | (swap & swap_mask)
|
| return atomic_response
```

The additional operands are carried in the Extended Transport Header. Atomic response generation and packet format for MskCmpSwap is as for standard IB Atomic operations.

3.2.6.1.2.2 Masked Fetch and Add (MFetchAdd)

The MFetchAdd Atomic operation extends the functionality of the standard IB FetchAdd by allowing the user to split the target into multiple fields of selectable length. The atomic add is done independently on each one of this fields. A bit set in the field_boundary parameter specifies the field boundaries. The pseudocode below describes the operation:

```
| bit_adder(ci, b1, b2, *co)
| {
|   value = ci + b1 + b2
```

```
      |   *co = !!(value & 2)
      |
      |   return value & 1
      | }
      |
      | #define MASK_IS_SET(mask, attr)      (!!((mask)&(attr)))
      | bit_position = 1
      | carry = 0
      | atomic_response = 0
      |
      | for i = 0 to 63
      | {
      |         if ( i != 0 )
      |                 bit_position =  bit_position << 1
      |
      |         bit_add_res = bit_adder(carry, MASK_IS_SET(*va, bit_position),
      |                                   MASK_IS_SET(compare_add, bit_position), &new_carry)
      |         if (bit_add_res)
      |                 atomic_response |= bit_position
      |
      |         carry = ((new_carry) && (!MASK_IS_SET(compare_add_mask, bit_position)))
      | }
      |
      | return atomic_response
```

3.2.6.2 XRC - eXtended Reliable Connected Transport Service for InfiniBand

XRC allows significant savings in the number of QPs and the associated memory resources required to establish all to all process connectivity in large clusters.

It significantly improves the scalability of the solution for large clusters of multicore end-nodes by reducing the required resources.

For further details, please refer to the "Annex A14 Supplement to InfiniBand Architecture Specification Volume 1.2.1"

A new API can be used by user space applications to work with the XRC transport. The legacy API is currently supported in both binary and source modes, however it is deprecated. Thus we recommend using the new API.

The new verbs to be used are:

- `ibv_open_xrcd/ibv_close_xrcd`
- `ibv_create_srq_ex`
- `ibv_get_srq_num`
- `ibv_create_qp_ex`
- `ibv_open_qp`

Please use ibv_xsrq_pingpong for basic tests and code reference. For detailed information regarding the various options for these verbs, please refer to their appropriate man pages.

3.2.6.3 Dynamically Connected Transport (DCT)

Dynamically Connected transport (DCT) service is an extension to transport services to enable a higher degree of scalability while maintaining high performance for sparse traffic. Utilization of DCT reduces the total number of QPs required system wide by having Reliable type QPs dynamically connect and disconnect from any remote node. DCT connections only stay connected while they are active. This results in smaller memory footprint, less overhead to set connections and higher on-chip cache utilization and hence increased performance. DCT is supported only in mlx5 driver.

3.2.7 Optimized Memory Access

3.2.7.1 Contiguous Pages

Contiguous Pages improves performance by allocating user memory regions over physical contiguous pages. It enables a user application to ask low level drivers to allocate contiguous memory for it as part of `ibv_reg_mr`.

Additional performance improvements can be reached by allocating Queue Pair (QP) and Completion Queue (CQ|) buffers to the Contiguous Pages.

To activate set the below environment variables with values of PREFER_CONTIG or CONTIG.

- For QP: MLX_QP_ALLOC_TYPE
- For CQ: MLX_CQ_ALLOC_TYPE

The following are all the possible values that can be allocated to the buffer:

Possible Value[1]	Description
ANON	Use current pages ANON small ones.
HUGE	Force huge pages.
CONTIG	Force contiguous pages.
PREFER_CONTIG	Try contiguous fallback to ANON small pages. (Default)
PREFER_HUGE	Try huge fallback to ANON small pages.
ALL	Try huge fallback to contiguous if failed fallback to ANON small pages.

1. Values are NOT case sensitive.

Usage:

The application calls the `ibv_exp_reg_mr` API which turns on the `IBV_EXP_ACCESS_ALLOCATE_MR` bit and sets the input address to NULL. Upon success, the address field of the struct `ibv_mr` will hold the address to the allocated memory block. This block will be freed implicitly when the `ibv_dereg_mr()` is called.

The following are environment variables that can be used to control error cases / contiguity:

Parameters	Description
`MLX_MR_ALLOC_TYPE`	Configures the allocator type. • `ALL` (Default) - Uses all possible allocator and selects most efficient allocator. • `ANON` - Enables the usage of anonymous pages and disables the allocator • `CONTIG` - Forces the usage of the contiguous pages allocator. If contiguous pages are not available the allocation fails
`MLX_MR_MAX_LOG2_CONTIG_BSIZE`	Sets the maximum contiguous block size order. • Values: 12-23 • Default: 23
`MLX_MR_MIN_LOG2_CONTIG_BSIZE`	Sets the minimum contiguous block size order. • Values: 12-23 • Default: 12

3.2.7.2 Shared Memory Region

Shared Memory Region is only applicable to the mlx4 driver.

Shared Memory Region (MR) enables sharing MR among applications by implementing the "Register Shared MR" verb which is part of the IB spec.

Sharing MR involves the following steps:

Step 1. Request to create a shared MR

The application sends a request via the `ibv_exp_reg_mr` API to create a shared MR. The application supplies the allowed sharing access to that MR. If the MR was created successfully, a unique MR ID is returned as part of the struct `ibv_mr` which can be used by other applications to register with that MR.

The underlying physical pages must not be Least Recently Used (LRU) or Anonymous. To disable that, you need to turn on the `IBV_EXP_ACCESS_ALLOCATE_MR` bit as part of the sharing bits.

Usage:

- Turns on via the `ibv_exp_reg_mr` one or more of the sharing access bits. The sharing bits are part of the `ibv_exp_reg_mr` man page.
- Turns on the `IBV_EXP_ACCESS_ALLOCATE_MR` bit

Step 2. Request to register to a shared MR

A new verb called `ibv_exp_reg_shared_mr` is added to enable sharing an MR. To use this verb, the application supplies the MR ID that it wants to register for and the desired access mode to that MR. The desired access is validated against its given permissions and upon successful creation, the physical pages of the original MR are shared by the new MR. Once the MR is shared, it can be used even if the original MR was destroyed.

The request to share the MR can be repeated multiple times and an arbitrary number of Memory Regions can potentially share the same physical memory locations.

Usage:

- Uses the "handle" field that was returned from the `ibv_exp_reg_mr` as the `mr_handle`
- Supplies the desired "access mode" for that MR
- Supplies the address field which can be either NULL or any hint as the required output. The address and its length are returned as part of the `ibv_mr struct`.

  To achieve high performance it is highly recommended to supply an address that is aligned as the original memory region address. Generally, it may be an alignment to 4M address.

For further information on how to use the `ibv_exp_reg_shared_mr` verb, please refer to the `ibv_exp_reg_shared_mr` man page and/or to the `ibv_shared_mr` sample program which demonstrates a basic usage of this verb.
Further information on the `ibv_shared_mr` sample program can be found in the `ibv_shared_mr` man page.

3.2.7.3 Memory Region Re-registration

Memory Region Re-registration allows the user to change attributes of the memory region. The user may change the PD, access flags or the address and length of the memory region. Memory region supports contagious pages allocation. Consequently, it de-registers memory region followed by register memory region. Where possible, resources are reused instead of de-allocated and reallocated.

Please note that the verb is implemented as an experimental verb.

```
int ibv_exp_rereg_mr(struct ibv_mr *mr, int flags, struct ibv_pd *pd, void *addr, size_t
length, uint64_t access, struct ibv_exp_rereg_mr_attr *attr);
```

```
@mr:                      The memory region to modify.
@flags:                   A bit-mask used to indicate which of the following properties
                          of the memory region are being modified. Flags should be one
                          of:
                          IBV_EXP_REREG_MR_CHANGE_TRANSLATION /* Change translation
                          (location and length) */                        IBV_EXP_RE-
                          REG_MR_CHANGE_PD/* Change protection domain*/
                          IBV_EXP_REREG_MR_CHANGE_ACCESS/* Change access flags*/
@pd:                      If IBV_EXP_REREG_MR_CHANGE_PD is set in flags, this field spec-
                          ifies the new protection domain to associated with the memory
                          region, otherwise, this parameter is ignored.
@addr:                    If IBV_EXP_REREG_MR_CHANGE_TRANSLATION is set in flags, this
                          field specifies the start of the virtual address to use in the
                          new translation, otherwise, this parameter is ignored.
@length:                  If IBV_EXP_REREG_MR_CHANGE_TRANSLATION is set in flags, this
                          field specifies the length of the virtual address to use in the
                          new translation, otherwise, this parameter is ignored.
```

```
@access:                  If IBV_EXP_REREG_MR_CHANGE_ACCESS is set in flags, this field
                          specifies the new memory access rights, otherwise, this param-
                          eter is ignored. Could be one of the following:
                          IBV_ACCESS_LOCAL_WRITE
                          IBV_ACCESS_REMOTE_WRITE
                          IBV_ACCESS_REMOTE_READ
                          IBV_ACCESS_ALLOCATE_MR      /* Let the library allocate the
                          memory for * the user, tries to get contiguous pages */
@attr:                    Future extensions
```

ibv_exp_rereg_mr returns 0 on success, or the value of an errno on failure (which indicates the error reason). In case of an error, the MR is in undefined state. The user needs to call ibv_-dereg_mr in order to release it.

Please note that if the MR (Memory Region) is created as a Shared MR and a translation is requested, after the call, the MR is no longer a shared MR. Moreover, Re-registration of MRs that uses Mellanox PeerDirect™ technology are not supported.

3.2.7.4 Memory Window

Memory Window allows the application to have a more flexible control over remote access to its memory. It is available only on physical functions / native machines The two types of Memory Windows supported are: type 1 and type 2B.

Memory Windows are intended for situations where the application wants to:

- grant and revoke remote access rights to a registered region in a dynamic fashion with less of a performance penalty
- grant different remote access rights to different remote agents and/or grant those rights over different ranges within registered region

For further information, please refer to the InfiniBand specification document.

Memory Windows API cannot co-work with peer memory clients (Mellanox PeerDirect™).

3.2.7.4.1 Query Capabilities

Memory Windows are available if and only the hardware supports it. To verify whether Memory Windows are available, run `ibv_exp_query_device`.

For example:

```
truct ibv_exp_device_attr device_attr = {.comp_mask = IBV_EXP_DEVICE_ATTR_RESERVED -
1};
ibv_exp_query_device(context, & device_attr);
if (device_attr.exp_device_cap_flags & IBV_EXP_DEVICE_MEM_WINDOW ||
                 device_attr.exp_device_cap_flags & IBV_EXP_DEVICE_MW_TYPE_2B) {
/* Memory window is supported */
```

3.2.7.4.2 Allocating Memory Window

Allocating memory window is done by calling the `ibv_alloc_mw` verb.

```
type_mw = IBV_MW_TYPE_2/ IBV_MW_TYPE_1
mw = ibv_alloc_mw(pd, type_mw);
```

3.2.7.4.3 Binding Memory Windows

After allocated, memory window should be bound to a registered memory region. Memory Region should have been registered using the `IBV_EXP_ACCESS_MW_BIND` access flag.

- Binding Memory Window **type 1** is done via the `ibv_exp_bind_mw` verb.

```
struct ibv_exp_mw_bind mw_bind = { .comp_mask = IBV_EXP_BIND_MW_RESERVED - 1 };
ret = ibv_exp_bind_mw(qp, mw, &mw_bind);
```

- Binding memory window **type 2B** is done via the `ibv_exp_post_send` verb and a specific Work Request (WR) with `opcode = IBV_EXP_WR_BIND_MW`

Prior to binding, please make sure to update the existing rkey.

```
ibv_inc_rkey(mw->rkey)
```

3.2.7.4.4 Invalidating Memory Window

Before rebinding Memory Window type 2, it must be invalidated using the `ibv_exp_post_send` verb and a specific WR with `opcode = IBV_EXP_WR_LOCAL_INV`.

3.2.7.4.5 Deallocating Memory Window

Deallocating memory window is done using the `ibv_dealloc_mw` verb.

```
ibv_dealloc_mw(mw);
```

3.2.7.5 User-Mode Memory Registration (UMR)

User-mode Memory Registration (UMR) is currently at beta level.

User-mode Memory Registration (UMR) is a fast registration mode which uses send queue. The UMR support enables the usage of RDMA operations and scatters the data at the remote side through the definition of appropriate memory keys on the remote side.

UMR enables the user to:

- Create indirect memory keys from previously registered memory regions, including creation of KLM's from previous KLM's. There are not data alignment or length restrictions associated with the memory regions used to define the new KLM's.
- Create memory regions, which support the definition of regular non-contiguous memory regions.

3.2.7.6 On-Demand-Paging (ODP)

On-Demand-Paging (ODP) is a technique to alleviate much of the shortcomings of memory registration. Applications no longer need to pin down the underlying physical pages of the address space, and track the validity of the mappings. Rather, the HCA requests the latest translations from the OS when pages are not present, and the OS invalidates translations which are no longer valid due to either non-present pages or mapping changes. ODP does not support the following features:

- Memory windows
- Indirect MKEY
- Contiguous pages

ODP can be further divided into 2 subclasses: Explicit and Implicit ODP.

- Explicit ODP

  In Explicit ODP, applications still register memory buffers for communication, but this operation is used to define access control for IO rather than pin-down the pages. ODP Memory Region (MR) does not need to have valid mappings at registration time.

- Implicit ODP

  In Implicit ODP, applications are provided with a special memory key that represents their complete address space. This all IO accesses referencing this key (subject to the access rights associated with the key) does not need to register any virtual address range.

3.2.7.6.1 Query Capabilities

On-Demand Paging is available if both the hardware and the kernel support it. To verify whether ODP is supported, run `ibv_exp_query_device`:

```
struct ibv_exp_device_attr dattr;
dattr.comp_mask = IBV_EXP_DEVICE_ATTR_ODP | IBV_EXP_DEVICE_ATTR_EXP_CAP_FLAGS;
ret = ibv_exp_query_device(context, &dattr);
if (dattr.exp_device_cap_flags  & IBV_EXP_DEVICE_ODP)
          //On-Demand Paging is supported.
```

Each transport has a capability field in the `dattr.odp_caps` structure that indicates which operations are supported by the ODP MR:

```
struct ibv_exp_odp_caps {
        uint64_t        general_odp_caps;
        struct {
                uint32_t        rc_odp_caps;
                uint32_t        uc_odp_caps;
                uint32_t        ud_odp_caps;
                uint32_t        dc_odp_caps;
                uint32_t        xrc_odp_caps;
                uint32_t        raw_eth_odp_caps;
        } per_transport_caps;
};
```

To check which operations are supported for a given transport, the capabilities field need to be masked with one of the following masks:

```
enum ibv_odp_transport_cap_bits {
        IBV_EXP_ODP_SUPPORT_SEND     = 1 << 0,
        IBV_EXP_ODP_SUPPORT_RECV     = 1 << 1,
        IBV_EXP_ODP_SUPPORT_WRITE    = 1 << 2,
        IBV_EXP_ODP_SUPPORT_READ     = 1 << 3,
        IBV_EXP_ODP_SUPPORT_ATOMIC   = 1 << 4,
        IBV_EXP_ODP_SUPPORT_SRQ_RECV = 1 << 5,
};
```

For example to check if RC supports send:

```
If (dattr.odp_caps.per_transport_caps.rc_odp_caps & IBV_EXP_ODP_SUPPORT_SEND)
                  //RC supports send operations with ODP MR
```

For further information, please refer to the `ibv_exp_query_device` manual page.

3.2.7.6.2 Registering ODP Explicit MR

ODP Explicit MR is registered after allocating the necessary resources (e.g., PD, buffer):

```
struct ibv_exp_reg_mr_in in;
struct ibv_mr *mr;
in.pd = pd;
in.addr = buf;
in.length = size;
in.exp_access = IBV_EXP_ACCESS_ON_DEMAND| … ;
in.comp_mask = 0;
mr = ibv_exp_reg_mr(&in);
```

Please be aware that the `exp_access` differs from one operation to the other, but the `IBV_EXP_-ACCESS_ON_DEMAND` is set for all ODP MRs.

For further information, please refer to the `ibv_exp_reg_mr` manual page.

3.2.7.6.3 Registering ODP Implicit MR

Registering an Implicit ODP MR provides you with an implicit lkey that represents the complete address space. Implicit ODP MR is limited to local access permissions (local read or write). It only has a valid lkey, whereas its rkey is invalid. The size of operations that can use this lkey is limited to 128MB.

To register an Implicit ODP MR, in addition to the `IBV_EXP_ACCESS_ON_DEMAND` access flag, use `in->addr = 0 and in->length = IBV_EXP_IMPLICIT_MR_SIZE.`

For further information, please refer to the `ibv_exp_reg_mr` manual page.

3.2.7.6.4 De-registering ODP MR

ODP MR is deregistered the same way a regular MR is deregistered:

```
ibv_dereg_mr(mr);
```

3.2.7.6.5 Pre-fetching Verb

The driver can pre-fetch a given range of pages and map them for access from the HCA. The pre-fetched verb is applicable for ODP MRs only, and it is done on a best effort basis, and may silently ignore errors.

Example:

```
struct ibv_exp_prefetch_attr prefetch_attr;
prefetch_attr.flags = IBV_EXP_PREFETCH_WRITE_ACCESS;
prefetch_attr.addr = addr;
prefetch_attr.length = length;
prefetch_attr.comp_mask = 0;
ibv_exp_prefetch_mr(mr, &prefetch_attr);
```

For further information, please refer to the `ibv_exp_prefetch_mr` manual page.

3.2.7.6.6 ODP Statistics

To aid in debugging and performance measurements and tuning, ODP support includes an extensive set of statistics. The statistics are divided into 2 sets: standard statistics and debug statistics. Both sets are maintained on a per-device basis and report the total number of events since the device was registered.

The standard statistics are reported as sysfs entries with the following format:

```
/sys/class/infiniband_verbs/uverbs[0/1]/
        invalidations_faults_contentions
        num_invalidation_pages
        num_invalidations
        num_page_fault_pages
        num_page_faults
        num_prefetchs_handled
        num_prefetch_pages
```

Counter Name	Description
invalidations_faults_contentions	Number of times that page fault events were dropped or prefetch operations were restarted due to OS page invalidations.
num_invalidation_pages	Total number of pages invalidated during all invalidation events.
num_invalidations	Number of invalidation events.
num_page_fault_pages	Total number of pages faulted in by page fault events.
num_page_faults	Number of page fault events.
num_prefetches_handled	Number of prefetch verb calls that were completed successfully.
num_prefetch_pages	Total number of pages that were prefetched by the prefetch verb.

The debug statistics are reported by debugfs entries with the following format:

```
/sys/kernel/debug/mlx5/<pci-dev-id>/odp_stats/
        num_failed_resolutions
        num_mrs_not_found
        num_odp_mr_pages
        num_odp_mrs
```

Counter Name	Description
num_failed_resolutions	Number of failed page faults that could not be resolved due to non-existing mappings in the OS.
num_mrs_not_found	Number of faults that specified a non-existing ODP MR.
num_odp_mr_pages	Total size in pages of current ODP MRs.
num_odp_mrs	Number of current ODP MRs.

3.2.7.7 Inline-Receive

When Inline-Receive is active, the HCA may write received data in to the receive WQE or CQE. Using Inline-Receive saves PCIe read transaction since the HCA does not need to read the scatter list, therefore it improves performance in case of short receive-messages.

On poll CQ, the driver copies the received data from WQE/CQE to the user's buffers. Therefore, apart from querying Inline-Receive capability and Inline-Receive activation the feature is transparent to user application.

When Inline-Receive is active, user application must provide a valid virtual address for the receive buffers to allow the driver moving the inline-received message to these buffers. The validity of these addresses is not checked therefore the result of providing non-valid virtual addresses is unexpected.

Connect-IB® supports Inline-Receive on both the requestor and the responder sides. Since data is copied at the poll CQ verb, Inline-Receive on the requestor side is possible only if the user chooses IB(V)_SIGNAL_ALL_WR.

3.2.7.7.1 Querying Inline-Receive Capability

User application can use the `ibv_exp_query_device` function to get the maximum possible Inline-Receive size. To get the size, the application needs to set the `IBV_EXP_DEVICE_ATTR_INLINE_RECV_SZ` bit in the `ibv_exp_device_attr comp_mask`.

3.2.7.7.2 Activating Inline-Receive

To activate the Inline-Receive, you need to set the required message size in the `max_inl_recv` field in the `ibv_exp_qp_init_attr struct` when calling `ibv_exp_create_qp` function. The value returned by the same field is the actual Inline-Receive size applied.

Setting the message size may affect the WQE/CQE size.

3.2.8 Mellanox PeerDirect™

Mellanox PeerDirect™ uses an API between IB CORE and peer memory clients, (e.g. GPU cards) to provide access to an HCA to read/write peer memory for data buffers. As a result, it allows RDMA-based (over InfiniBand/RoCE) application to use peer device computing power, and RDMA interconnect at the same time without copying the data between the P2P devices.

For example, Mellanox PeerDirect™ is being used for GPUDirect RDMA.

Detailed description for that API exists under MLNX OFED installation, please see docs/readme_and_user_manual/PEER_MEMORY_API.txt

3.2.9 CPU Overhead Distribution

When creating a CQ using the `ibv_create_cq()` API, a `"comp_vector"` argument is sent. If the value set for this argument is 0, while the CPU core executing this verb is not equal to zero, the driver assigns a completion EQ with the least CQs reporting to it. This method is used to distribute CQs amongst available completions EQ. To assign a CQ to a specific EQ, the EQ needs to be specified in the `comp_vector` argument.

3.2.10 Resource Domain Experimental Verbs

Resource domain is a verb object which may be associated with QP and a CQ objects on creation to enhance data-path performance.

3.2.10.1 Resource Domain Attributes

Thread model - Defines the threading module for the objects (QP/CQ) associated with this resource domain.

`IBV_EXP_THREAD_SAFE`	Access to the associated objects are thread safe. Such objects may be accessed by any thread without concern for thread safety issues.
`IBV_EXP_THREAD_UNSAFE`	Access to the associated objects are not thread safe. Access to such objects must be coordinated by the calling threads.
`IBV_EXP_THREAD_SINGLE`	Access to the associated objects are from only one thread over the lifetime of the object. In addition, different objects associated with the same resource domain must be called by the same thread.

Message model - Defines whether associated objects connection is optimized for low latency or high bandwidth.

`IBV_EXP_MSG_FORCE_LOW_LATENCY`	Forces the provider to optimize for low latency
`IBV_EXP_MSG_LOW_LATENCY`	Suggests the provider to optimize for low latency.
`IBV_EXP_MSG_HIGH_BW`	Suggests the provider to optimize for high bandwidth.
`IBV_EXP_MSG_DEFAULT`	Uses the provider’s default message model.

➢ ***To create resource domain use:***

```
static inline struct ibv_exp_res_domain *ibv_exp_create_res_domain(
struct ibv_context *context,  struct ibv_exp_res_domain_init_attr *attr)
```

➢ ***To destroy resource domain use:***

```
static inline int ibv_exp_destroy_res_domain( struct ibv_context *context,
struct ibv_exp_res_domain *res_dom, struct ibv_exp_destroy_res_domain_attr *attr)
```

Use the res_domain field and relevant comp_mask in the ibv_exp_cq_init_attr and ibv_exp_qp_init_attr structs to pass resource domain to the CQ and QP on their creation.

Example of creating CQ and QP which may be called from only one thread:

```
struct ibv_exp_res_domain_init_attr res_domain_attr;
struct ibv_exp_res_domain *res_domain;
struct ibv_exp_cq_init_attr cq_attr;
struct ibv_exp_qp_init_attr qp_attr;

…
res_domain_attr.comp_mask = IBV_EXP_RES_DOMAIN_THREAD_MODEL |
IBV_EXP_RES_DOMAIN_MSG_MODEL;
res_domain_attr.thread_model = IBV_EXP_THREAD_SINGLE;
res_domain_attr.msg_model = IBV_EXP_MSG_HIGH_BW;
```

```
res_domain = ibv_exp_create_res_domain(ctx->context, &res_domain_attr);
if (!_domain) {
     fprintf(stderr, "Can't create resource domain\n");
     exit(1);
}
…
cq_attr.res_domain = res_domain;
cq_attr.comp_mask |= IBV_EXP_CQ_INIT_ATTR_RES_DOMAIN;
cq = ibv_exp_create_cq(context, num_ entries, NULL, NULL, 0, &cq_attr);
…
qp_attr.res_domain = res_domain;
qp_attr.comp_mask |= IBV_EXP_QP_INIT_ATTR_RES_DOMAIN;
qp = ibv_exp_create_qp(context, &qp_attr);
```

3.2.11 Query Interface Experimental Verbs

The new experimental verbs `ibv_exp_query_intf` and `ibv_exp_release_intf` provide a mechanism to extend the verbs with families of verbs interfaces. These extensions provide a way to optimize data-path interfaces (e.g. `post-send/recv, poll-cq`) for specific applications (e.g. DPDK).

The interfaces families provided by the new verbs may be vendor specific families (in this case, the vendor should provide the header file for the interface definition) or global families which will be defined in `verbs.h`.

For more information regarding the new verbs, please refer to their man pages and the `ibv_intf` example.

3.2.11.1 QP-burst Experimental Family

The `ibv_exp_qp_burst_family` is an extension interface to the legacy `post_send/receive` verbs. This family provides an interface for applications that need fast `send/recv` messages (e.g. DPDK).

To get an instance of the `ibv_exp_qp_burst_family`, the application needs to use the query-interface mechanism (`ibv_exp_query_intf`).

For more information regarding the i`bv_exp_qp_burst_family`, please refer to its man page and the `ibv_intf` example.

3.2.11.2 CQ Experimental Family

The `ibv_exp_cq_family` is an extension interface to the legacy `poll_cq` verb. This family provides an interface for applications that need fast `send/recv` messages (e.g. DPDK).

To get an instance of the `ibv_exp_cq_family` the application needs to use the query-interface mechanism (`ibv_exp_query_intf`).

For more information regarding the `ibv_exp_cq_family`, please refer to its man page and the `ibv_intf` example.

3.2.11.3 WQ Experimental Family

The `ibv_exp_wq_family` provides interface to post receive WR to the receive WQ. This family provides an interface for applications that need fast send/recv messages (e.g. DPDK).

To get an instance of the `ibv_exp_wq_family` the application needs to use the query-interface mechanism (`ibv_exp_query_intf`).

For more information regarding the `ibv_exp_wq_family`, please refer to `ibv_exp_query_intf` man page.

3.2.12 WQE Format in MLNX_OFED

Please refer to section Section 3.1.12, "WQE Format in MLNX_OFED", on page 84.

3.3 Storage Protocols

3.3.1 SCSI RDMA Protocol (SRP)

As described in Section 3.3.1, the SCSI RDMA Protocol (SRP) is designed to take full advantage of the protocol off-load and RDMA features provided by the InfiniBand architecture. SRP allows a large body of SCSI software to be readily used on InfiniBand architecture. The SRP Initiator controls the connection to an SRP Target in order to provide access to remote storage devices across an InfiniBand fabric. The kSRP Target resides in an IO unit and provides storage services.

Section 3.3.1.1 describes the SRP Initiator included in Mellanox OFED for Linux. This package, however, does *not* include an SRP Target.

3.3.1.1 SRP Initiator

This SRP Initiator is based on open source from OpenFabrics (www.openfabrics.org) that implements the SCSI RDMA Protocol-2 (SRP-2). SRP-2 is described in Document # T10/1524-D available from http://www.t10.org.

The SRP Initiator supports

- Basic SCSI Primary Commands -3 (SPC-3)

  (www.t10.org/ftp/t10/drafts/spc3/spc3r21b.pdf)
- Basic SCSI Block Commands -2 (SBC-2)

  (www.t10.org/ftp/t10/drafts/sbc2/sbc2r16.pdf)
- Basic functionality, task management and limited error handling

Loading SRP Initiator

To load the SRP module, either execute the "`modprobe ib_srp`" command after the OFED driver is up, or change the value of SRP_LOAD in `/etc/infiniband/openib.conf` to "yes".

For the changes to take effect, run: /etc/init.d/openibd restart

When loading the ib_srp module, it is possible to set the module parameter srp_s-g_tablesize. This is the maximum number of gather/scatter entries per I/O (default: 12).

3.3.1.1.1 SRP Module Parameters

When loading the SRP module, the following parameters can be set (viewable by the `"modinfo ib_srp"` command):

cmd_sg_entries	Default number of gather/scatter entries in the SRP command (default is 12, max 255)
allow_ext_sg	Default behavior when there are more than cmd_sg_entries S/G entries after mapping; fails the request when false (default false)
topspin_workarounds	Enable workarounds for Topspin/Cisco SRP target bugs
reconnect_delay	Time between successive reconnect attempts. Time between successive reconnect attempts of SRP initiator to a disconnected target until dev_loss_tmo timer expires (if enabled), after that the SCSI target will be removed.
fast_io_fail_tmo	Number of seconds between the observation of a transport layer error and failing all I/O. Increasing this timeout allows more tolerance to transport errors, however, doing so increases the total failover time in case of serious transport failure. Note: fast_io_fail_tmo value must be smaller than the value of reconnect_delay.
dev_loss_tmo	Maximum number of seconds that the SRP transport should insulate transport layer errors. After this time has been exceeded the SCSI target is removed. Normally it is advised to set this to -1 (disabled) which will never remove the scsi_host. In deployments where different SRP targets are connected and disconnected frequently, it may be required to enable this timeout in order to clean old scsi_hosts representing targets that no longer exists.

Constraints between parameters:

- `dev_loss_tmo, fast_io_fail_tmo, reconnect_delay` cannot be all disabled or negative values.
- `reconnect_delay` must be positive number.
- `fast_io_fail_tmo` must be smaller than SCSI block device timeout.
- `fast_io_fail_tmo` must be smaller than `dev_loss_tmo`.

3.3.1.1.2 SRP Remote Ports Parameters

Several SRP remote ports parameters are modifiable online on existing connection.

➢ ***To modify dev_loss_tmo to 600 seconds:***

```
echo 600 > /sys/class/srp_remote_ports/port-xxx/dev_loss_tmo
```

➢ ***To modify fast_io_fail_tmo to 15 seconds:***

```
echo 15 > /sys/class/srp_remote_ports/port-xxx/fast_io_fail_tmo
```

➢ ***To modify reconnect_delay to 10 seconds:***

```
echo 20 > /sys/class/srp_remote_ports/port-xxx/reconnect_delay
```

Manually Establishing an SRP Connection

The following steps describe how to manually load an SRP connection between the Initiator and an SRP Target. Section , “Automatic Discovery and Connection to Targets”, on page 184 explains how to do this automatically.

- Make sure that the `ib_srp` module is loaded, the SRP Initiator is reachable by the SRP Target, and that an SM is running.
- To establish a connection with an SRP Target and create an SRP (SCSI) device for that target under `/dev`, use the following command:

  ```
  echo -n id_ext=[GUID value],ioc_guid=[GUID value],dgid=[port GID value],\
  pkey=ffff,service_id=[service[0] value] > \
  /sys/class/infiniband_srp/srp-mlx[hca number]-[port number]/add_target
  ```

  See Section , “SRP Tools - ibsrpdm, srp_daemon and srpd Service Script”, on page 182 for instructions on how the parameters in this `echo` command may be obtained.

  Notes:

  - Execution of the above “echo” command may take some time
  - The SM must be running while the command executes
  - It is possible to include additional parameters in the echo command:
    - max_cmd_per_lun - Default: 62
    - max_sect (short for max_sectors) - sets the request size of a command
    - io_class - Default: 0x100 as in rev 16A of the specification (In rev 10 the default was 0xff00)
    - tl_retry_count - a number in the range 2..7 specifying the IB RC retry count. Default: 2
    - comp_vector, a number in the range 0..n-1 specifying the MSI-X completion vector. Some HCA's allocate multiple (n) MSI-X vectors per HCA port. If the IRQ affinity masks of these interrupts have been configured such that each MSI-X interrupt is handled by a different CPU then the comp_vector parameter can be used to spread the SRP completion workload over multiple CPU's.
    - cmd_sg_entries, a number in the range 1..255 that specifies the maximum number of data buffer descriptors stored in the SRP_CMD information unit itself. With allow_ext_sg=0 the parameter cmd_sg_entries defines the maximum S/G list length for a single SRP_CMD, and commands whose S/G list length exceeds this limit after S/G list collapsing will fail.
    - initiator_ext - Please refer to Section 9 (Multiple Connections...)
- To list the new SCSI devices that have been added by the echo command, you may use either of the following two methods:
  - Execute “fdisk -l”. This command lists all devices; the new devices are included in this listing.
  - Execute “dmesg” or look at `/var/log/messages` to find messages with the names of the new devices.

3.3.1.1.3 SRP sysfs Parameters

Interface for making ib_srp connect to a new target. One can request ib_srp to connect to a new target by writing a comma-separated list of login parameters to this sysfs attribute. The supported parameters are:

id_ext	A 16-digit hexadecimal number specifying the eight byte identifier extension in the 16-byte SRP target port identifier. The target port identifier is sent by ib_srp to the target in the SRP_LOGIN_REQ request.

ioc_guid	A 16-digit hexadecimal number specifying the eight byte I/O controller GUID portion of the 16-byte target port identifier.
dgid	A 32-digit hexadecimal number specifying the destination GID.
pkey	A four-digit hexadecimal number specifying the InfiniBand partition key.
service_id	A 16-digit hexadecimal number specifying the InfiniBand service ID used to establish communication with the SRP target. How to find out the value of the service ID is specified in the documentation of the SRP target.
max_sect	A decimal number specifying the maximum number of 512-byte sectors to be transferred via a single SCSI command.
max_cmd_per_lun	A decimal number specifying the maximum number of outstanding commands for a single LUN.
io_class	A hexadecimal number specifying the SRP I/O class. Must be either 0xff00 (rev 10) or 0x0100 (rev 16a). The I/O class defines the format of the SRP initiator and target port identifiers.
initiator_ext	A 16-digit hexadecimal number specifying the identifier extension portion of the SRP initiator port identifier. This data is sent by the initiator to the target in the SRP_LOGIN_REQ request.
cmd_sg_entries	A number in the range 1..255 that specifies the maximum number of data buffer descriptors stored in the SRP_CMD information unit itself. With allow_ext_sg=0 the parameter cmd_sg_entries defines the maximum S/G list length for a single SRP_CMD, and commands whose S/G list length exceeds this limit after S/G list collapsing will fail.
allow_ext_sg	Whether ib_srp is allowed to include a partial memory descriptor list in an SRP_CMD instead of the entire list. If a partial memory descriptor list has been included in an SRP_CMD the remaining memory descriptors are communicated from initiator to target via an additional RDMA transfer. Setting allow_ext_sg to 1 increases the maximum amount of data that can be transferred between initiator and target via a single SCSI command. Since not all SRP target implementations support partial memory descriptor lists the default value for this option is 0.
sg_tablesize	A number in the range 1..2048 specifying the maximum S/G list length the SCSI layer is allowed to pass to ib_srp. Specifying a value that exceeds cmd_sg_entries is only safe with partial memory descriptor list support enabled (allow_ext_sg=1).
comp_vector	A number in the range 0..n-1 specifying the MSI-X completion vector. Some HCA's allocate multiple (n) MSI-X vectors per HCA port. If the IRQ affinity masks of these interrupts have been configured such that each MSI-X interrupt is handled by a different CPU then the comp_vector parameter can be used to spread the SRP completion workload over multiple CPU's.
tl_retry_count	A number in the range 2..7 specifying the IB RC retry count.

SRP Tools - ibsrpdm, srp_daemon and srpd Service Script

To assist in performing the steps in Section 6, the OFED distribution provides two utilities, ibsrpdm and srp_daemon, which

- Detect targets on the fabric reachable by the Initiator (for Step 1)
- Output target attributes in a format suitable for use in the above “echo” command (Step 2)
- A service script srpd which may be started at stack startup

The utilities can be found under `/usr/sbin/`, and are part of the srptools RPM that may be installed using the Mellanox OFED installation. Detailed information regarding the various options for these utilities are provided by their man pages.

Below, several usage scenarios for these utilities are presented.

3.3.1.1.4 ibsrpdm

ibsrpdm is using for the following tasks:

1. Detecting reachable targets
   a. To detect all targets reachable by the SRP initiator via the default umad device (/sys/class/infiniband_mad/umad0), execute the following command:

```
ibsrpdm
```

   This command will output information on each SRP Target detected, in human-readable form.

   Sample output:

```
    IO Unit Info:
         port LID:        0103
         port GID:        fe800000000000000002c90200402bd5
         change ID:       0002
         max controllers: 0x10
      controller[  1]
          GUID:      0002c90200402bd4
          vendor ID: 0002c9
          device ID: 005a44
          IO class : 0100
          ID:        LSI Storage Systems SRP Driver 200400a0b81146a1
          service entries: 1
          service[  0]: 200400a0b81146a1 / SRP.T10:200400A0B81146A1
```

   b. To detect all the SRP Targets reachable by the SRP Initiator via another umad device, use the following command:

```
ibsrpdm -d <umad device>
```

2. Assistance in creating an SRP connection
   a. To generate output suitable for utilization in the “echo” command of Section , “Manually Establishing an SRP Connection”, on page 180, add the
   ‘-c’ option to ibsrpdm:

```
ibsrpdm -c
```

Sample output:

```
id_ext=200400A0B81146A1,ioc_guid=0002c90200402bd4,
dgid=fe800000000000000002c90200402bd5,pkey=ffff,service_id=200400a0b81146a1
```

b. To establish a connection with an SRP Target using the output from the '`ibsrpdm -c`' example above, execute the following command:

```
echo -n id_ext=200400A0B81146A1,ioc_guid=0002c90200402bd4,
dgid=fe800000000000000002c90200402bd5,pkey=ffff,service_id=200400a0b81146a1 > /sys/
class/infiniband_srp/srp-mlx4_0-1/add_target
```

The SRP connection should now be up; the newly created SCSI devices should appear in the listing obtained from the '`fdisk -l`' command.

3. Discover reachable SRP Targets given an InfiniBand HCA name and port, rather than by just runing `/sys/class/infiniband_mad/umad<N>` where <N> is a digit

3.3.1.1.5 srpd

The srpd service script allows automatic activation and termination of the srp_daemon utility on all system live InfiniBand ports.

3.3.1.1.6 srp_daemon

The srp_daemon utility is based on ibsrpdm and extends its functionality. In addition to the ibsrpdm functionality described above, srp_daemon can also

- Establish an SRP connection by itself (without the need to issue the "echo" command described in Section , "Manually Establishing an SRP Connection", on page 180)
- Continue running in background, detecting new targets and establishing SRP connections with them (daemon mode)
- Discover reachable SRP Targets given an infiniband HCA name and port, rather than just by
  /dev/umad<N> where <N> is a digit
- Enable High Availability operation (together with Device-Mapper Multipath)
- Have a configuration file that determines the targets to connect to

1. `srp_daemon` commands equivalent to ibsrpdm:

```
"srp_daemon -a -o"    is equivalent to "ibsrpdm"
"srp_daemon -c -a -o" is equivalent to "ibsrpdm -c"
```

These srp_daemon commands can behave differently than the equivalent ibsrpdm command when /etc/srp_daemon.conf is not empty.

2. `srp_daemon` extensions to `ibsrpdm`
   - To discover SRP Targets reachable from the HCA device <InfiniBand HCA name> and the port <port num>, (and to generate output suitable for 'echo',) you may execute:

```
host1# srp_daemon -c -a -o -i <InfiniBand HCA name> -p <port number>
```

To obtain the list of InfiniBand HCA device names, you can either use the ibstat tool or run 'ls /sys/class/infiniband'.

- To both discover the SRP Targets and establish connections with them, just add the -e option to the above command.
- Executing srp_daemon over a port without the -a option will only display the reachable targets via the port and to which the initiator is not connected. If executing with the -e option it is better to omit -a.
- It is recommended to use the -n option. This option adds the initiator_ext to the connecting string. (See Section for more details).
- srp_daemon has a configuration file that can be set, where the default is /etc/srp_daemon.conf. Use the -f to supply a different configuration file that configures the targets srp_daemon is allowed to connect to. The configuration file can also be used to set values for additional parameters (e.g., max_cmd_per_lun, max_sect).
- A continuous background (daemon) operation, providing an automatic ongoing detection and connection capability. See Section .

Automatic Discovery and Connection to Targets

- Make sure that the ib_srp module is loaded, the SRP Initiator can reach an SRP Target, and that an SM is running.
- To connect to all the existing Targets in the fabric, run "`srp_daemon -e -o`". This utility will scan the fabric once, connect to every Target it detects, and then exit.

srp_daemon will follow the configuration it finds in /etc/srp_daemon.conf. Thus, it will ignore a target that is disallowed in the configuration file.

- To connect to all the existing Targets in the fabric and to connect to new targets that will join the fabric, execute srp_daemon -e. This utility continues to execute until it is either killed by the user or encounters connection errors (such as no SM in the fabric).
- To execute SRP daemon as a daemon on all the ports:
  - `srp_daemon.sh` (found under /usr/sbin/). `srp_daemon.sh` sends its log to `/var/log/srp_daemon.log`.
  - Start the srpd service script, run `service srpd start`
- It is possible to configure this script to execute automatically when the InfiniBand driver starts by changing the value of SRP_DAEMON_ENABLE in /etc/infiniband/openib.conf to "yes". However, this option also enables SRP High Availability that has some more features – see Section , "High Availability (HA)", on page 185).

  For the changes in `openib.conf` to take effect, run:
  ```
  /etc/init.d/openibd restart
  ```

Multiple Connections from Initiator InfiniBand Port to the Target

Some system configurations may need multiple SRP connections from the SRP Initiator to the same SRP Target: to the same Target IB port, or to different IB ports on the same Target HCA.

In case of a single Target IB port, i.e., SRP connections use the same path, the configuration is enabled using a different initiator_ext value for each SRP connection. The initiator_ext value is a 16-hexadecimal-digit value specified in the connection command.

Also in case of two physical connections (i.e., network paths) from a single initiator IB port to two different IB ports on the same Target HCA, there is need for a different initiator_ext value on each path. The conventions is to use the Target port GUID as the initiator_ext value for the relevant path.

If you use srp_daemon with -n flag, it automatically assigns initiator_ext values according to this convention. For example:

```
id_ext=200500A0B81146A1,ioc_guid=0002c90200402bec,\
dgid=fe800000000000000002c90200402bed,pkey=ffff,\ service_id=200500a0b81146a1,initia-
tor_ext=ed2b400002c90200
```

Notes:

1. It is recommended to use the -n flag for all srp_daemon invocations.
2. ibsrpdm does not have a corresponding option.
3. srp_daemon.sh always uses the -n option (whether invoked manually by the user, or automatically at startup by setting SRP_DAEMON_ENABLE to yes).

High Availability (HA)

High Availability works using the Device-Mapper (DM) multipath and the SRP daemon. Each initiator is connected to the same target from several ports/HCAs. The DM multipath is responsible for joining together different paths to the same target and for fail-over between paths when one of them goes offline. Multipath will be executed on newly joined SCSI devices.

Each initiator should execute several instances of the SRP daemon, one for each port. At startup, each SRP daemon detects the SRP Targets in the fabric and sends requests to the ib_srp module to connect to each of them. These SRP daemons also detect targets that subsequently join the fabric, and send the ib_srp module requests to connect to them as well.

3.3.1.1.7 Operation

When a path (from port1) to a target fails, the ib_srp module starts an error recovery process. If this process gets to the reset_host stage and there is no path to the target from this port, ib_srp will remove this scsi_host. After the scsi_host is removed, multipath switches to another path to this target (from another port/HCA).

When the failed path recovers, it will be detected by the SRP daemon. The SRP daemon will then request ib_srp to connect to this target. Once the connection is up, there will be a new scsi_host for this target. Multipath will be executed on the devices of this host, returning to the original state (prior to the failed path).

3.3.1.1.8 Manual Activation of High Availability

Initialization: (Execute after each boot of the driver)

1. Execute modprobe dm-multipath
2. Execute modprobe ib-srp

3. Make sure you have created file /etc/udev/rules.d/91-srp.rules as described above.
4. Execute for each port and each HCA:

```
srp_daemon -c -e -R 300 -i <InfiniBand HCA name> -p <port number>
```

   This step can be performed by executing srp_daemon.sh, which sends its log to /var/log/srp_-daemon.log.

Now it is possible to access the SRP LUNs on `/dev/mapper/`.

It is possible for regular (non-SRP) LUNs to also be present; the SRP LUNs may be identified by their names. You can configure the /etc/multipath.conf file to change multipath behavior.

It is also possible that the SRP LUNs will not appear under /dev/mapper/. This can occur if the SRP LUNs are in the black-list of multipath. Edit the 'blacklist' section in /etc/multipath.conf and make sure the SRP LUNs are not black-listed.

3.3.1.1.9 Automatic Activation of High Availability

- Set the value of SRP_DAEMON_ENABLE in `/etc/infiniband/openib.conf` to "yes".

  For the changes in `openib.conf` to take effect, run:
  `/etc/init.d/openibd restart`
- Start srpd service, run: `service srpd start`
- From the next loading of the driver it will be possible to access the SRP LUNs on /dev/mapper/

It is possible that regular (not SRP) LUNs may also be present; the SRP LUNs may be identified by their name.

- It is possible to see the output of the SRP daemon in /var/log/srp_daemon.log

Shutting Down SRP

SRP can be shutdown by using "rmmod ib_srp", or by stopping the OFED driver ("`/etc/init.d/openibd stop`"), or as a by-product of a complete system shutdown.

Prior to shutting down SRP, remove all references to it. The actions you need to take depend on the way SRP was loaded. There are three cases:

1. Without High Availability

   When working without High Availability, you should unmount the SRP partitions that were mounted prior to shutting down SRP.
2. After Manual Activation of High Availability

If you manually activated SRP High Availability, perform the following steps:

   a. Unmount all SRP partitions that were mounted.
   b. Stop service srpd (Kill the SRP daemon instances).

c. Make sure there are no multipath instances running. If there are multiple instances, wait for them to end or kill them.

d. Run: `multipath -F`

3. After Automatic Activation of High Availability

   If SRP High Availability was automatically activated, SRP shutdown must be part of the driver shutdown ("/etc/init.d/openibd stop") which performs Steps 2-4 of case b above. However, you still have to unmount all SRP partitions that were mounted before driver shutdown.

3.3.2 Sockets Direct Protocol (SDP)

Sockets Direct Protocol (SDP) is an InfiniBand byte-stream transport protocol that provides TCP stream semantics. Capable of utilizing InfiniBand's advanced protocol offload capabilities, SDP can provide lower latency, higher bandwidth, and lower CPU utilization than IPoIB or Ethernet running some sockets-based applications.

SDP can be used by applications and improve their performance transparently (that is, without any re-compilation). Since SDP has the same socket semantics as TCP, an existing application is able to run using SDP; the difference is that the application's TCP socket gets replaced with an SDP socket.

It is also possible to configure the driver to automatically translate TCP to SDP based on the source IP/port, the destination, or the application name. See Section 3.3.2.4.

The SDP protocol is composed of a kernel module that implements the SDP as a new address-family/protocol-family, and a library (see Section 3.3.2.1) that is used for replacing the TCP address family with SDP according to a policy.

3.3.2.1 libsdp.so Library

`libsdp.so` is a dynamically linked library, which is used for transparent integration of applications with SDP. The library is preloaded, and therefore takes precedence over glibc for certain socket calls. Thus, it can transparently replace the TCP socket family with SDP socket calls.

The library also implements a user-level socket switch. Using a configuration file, the system administrator can set up the policy that selects the type of socket to be used. `libsdp.so` also has the option to allow server sockets to listen on both SDP and TCP interfaces. The various configurations with SDP/TCP sockets are explained inside the `/etc/libsdp.conf` file.

3.3.2.2 Configuring SDP

To load SDP upon boot, edit the file `/etc/infiniband/openib.conf` and set "SDP_-LOAD=yes".

For the changes to take effect, run: **`/etc/init.d/openibd restart`**

SDP can work over IPoIB interfaces or RoCE interfaces. In case of IPoIB, SDP uses the same IP addresses and interface names as IPoIB (see IPoIB configuration in Section 3.2.5.1.2 and Section 3.2.5.1.2.5). In case of RoCE, SDP use the same IP addresses and interface names of the corresponding mlx4_en interfaces.

3.3.2.2.1 How to Know SDP Is Working

Since SDP is a transparent TCP replacement, it can sometimes be difficult to know that it is working correctly. To check whether traffic is passing through SDP or TCP, monitor the file /proc/net/sdpstats and see which counters are running.

3.3.2.2.1.1 Alternative Method – Using the sdpnetstat Program

The `sdpnetstat` program can be used to verify both that SDP is loaded and is being used. The following command shows all active SDP sockets using the same format as the traditional `netstat` program. Without the '-S' option, it shows all the information that `netstat` does plus SDP data.

```
host1$ sdpnetstat -S
```

Assuming that the SDP kernel module is loaded and is being used, then the output of the command will be as follows:

```
host1$ sdpnetstat -S
Proto Recv-Q Send-Q Local Address           Foreign Address
sdp        0      0 193.168.10.144:34216    193.168.10.125:12865
sdp        0 884720 193.168.10.144:42724    193.168.10.:filenet-rmi
```

The example output above shows two active SDP sockets and contains details about the connections.

If the SDP kernel module is not loaded, then the output of the command will be something like the following:

```
host1$ sdpnetstat -S
Proto Recv-Q Send-Q Local Address           Foreign Address
netstat: no support for `AF INET (tcp)' on this system.
```

To verify whether the module is loaded or not, you can use the `lsmod` command:

```
ib_sdp1250200
```

The example output above shows that the SDP module is loaded.

If the SDP module *is* loaded and the `sdpnetstat` command did not show SDP sockets, then SDP is not being used by any application.

3.3.2.2.2 Monitoring and Troubleshooting Tools

SDP has debug support for both the user space `libsdp.so` library and the `ib_sdp` kernel module. Both can be useful to understand why a TCP socket was not redirected over SDP and to help find problems in the SDP implementation.

3.3.2.2.2.1 User Space SDP Debug

User-space SDP debug is controlled by options in the `libsdp.conf` file. You can also have a local version and point to it explicitly using the following command:

```
host1$ export LIBSDP_CONFIG_FILE=<path>/libsdp.conf
```

To obtain extensive debug information, you can modify `libsdp.conf` to have the `log` directive produce maximum debug output (provide the `min-level` flag with the value 1).

The **`log`** statement enables the user to specify the debug and error messages that are to be sent and their destination. The syntax of **`log`** is as follows:

```
log [destination (stderr | syslog | file <filename>)] [min-level 1-9]
```

where options are:

```
destination      send log messages to the specified destination:
                 stderr: forward messages to the STDERR
                 syslog: send messages to the syslog service
                 file <filename>: write messages to the file
                 /var/log/filename for root. For a regular
                 user, write to /tmp/<filename>.<uid> if filename is
                 not specified as a full path; otherwise, write to
                 <path>/<filename>.<uid>
min-level        verbosity level of the log:
                 9: print errors only
                 8: print warnings
                  7: print connect and listen summary (useful for tracking
                    SDP usage)
                 4: print positive match summary (useful for config file
                    debug)
                 3: print negative match summary (useful for config file
                    debug)
                 2: print function calls and return values
                 1: print debug messages
```

Examples:

To print SDP usage per connect and listern to STDERR, include the following statement:

```
log min-level 7 destination stderr
```

A non-root user can configure `libsdp.so` to record function calls and return values in the file `/tmp/libsdp.log.<pid>` (root log goes to `/var/log/libsdp.log` for this example) by including the following statement in `libsdp.conf`:

```
log min-level 2 destination file libsdp.log
```

To print errors only to syslog, include the following statement:

```
log min-level 9 destination syslog
```

To print maximum output to the file /tmp/sdp_debug.log.<pid>, include the following statement:

```
log min-level 1 destination file sdp_debug.log
```

3.3.2.2.2.2 Kernel Space SDP Debug

The SDP kernel module can log detailed trace information if you enable it using the 'debug_level' variable in the sysfs filesystem. The following command performs this:

```
host1$ echo 1 > /sys/module/ib_sdp/parameters/sdp_debug_level
```

Depending on the operating system distribution on your machine, you may need an extra level—parameters— in the directory structure, so you may need to direct the echo command to /sys/module/ib_sdp/parameters/debug_level.

Turning off kernel debug is done by setting the sysfs variable to zero using the following command:

```
host1$ echo 0 > /sys/module/ib_sdp/parameters/sdp_debug_level
```

To display debug information, use the dmesg command:

```
host1$ dmesg
```

3.3.2.3 Environment Variables

For the transparent integration with SDP, the following two environment variables are required:

1. LD_PRELOAD – this environment variable is used to preload libsdp.so and it should point to the libsdp.so library. The variable should be set by the system administrator to /usr/lib/libsdp.so (or /usr/lib64/libsdp.so).
2. LIBSDP_CONFIG_FILE – this environment variable is used to configure the policy for replacing TCP sockets with SDP sockets. By default it points to: /etc/libsdp.conf.
3. SIMPLE_LIBSDP – ignore libsdp.conf and always use SDP

3.3.2.4 Converting Socket-based Applications

You can convert a socket-based application to use SDP instead of TCP in an automatic (also called transparent) mode or in an explicit (also called non-transparent) mode.

3.3.2.4.1 Automatic (Transparent) Conversion

The libsdp.conf configuration (policy) file is used to control the automatic transparent replacement of TCP sockets with SDP sockets. In this mode, socket streams are converted based upon a destination port, a listening port, or a program name.

Socket control statements in libsdp.conf allow the user to specify when libsdp should replace AF_INET/SOCK_STREAM sockets with AF_SDP/SOCK_STREAM sockets. Each control statement specifies a matching rule that applies if all its subexpressions must evaluate as true (logical and).

The use statement controls which type of sockets to open. The format of a use statement is as follows:

```
use <address-family> <role> <program-name|*> <address|*>:<port range|*>
```

where

```
<address-family>
               can be one of
               sdp: for specifying when an SDP should be used
               tcp: for specifying when an SDP socket should not be
               matched
               both: for specifying when both SDP and AF_INET sockets
               should be used
               Note that both semantics is different for server and
                client roles. For server, it means that the server will
               be listening on both SDP and TCP sockets. For client,
               the connect function will first attempt to use SDP and
               will silently fall back to TCP if the SDP connection
               fails.
<role>
               can be one of
                 server or listen: for defining the listening port address
               family
               client or connect: for defining the connected port
               address family
<program-name|*>
                Defines the program name the rule applies to (not includ
                 ing the path). Wildcards with same semantics as 'ls' are
                supported (* and ?). So db2* would match on any program
                 with a name starting with db2. t?cp would match on ttcp,
                etc.
                If program-name is not provided (default), the statement
                matches all programs.
<address|*>
                Either the local address to which the server binds,
                or the remote server address to which the client
                connects. The syntax for address matching is:
                <IPv4 address>[/<prefix_length>]|*
                IPv4 address = [0-9]+\.[0-9]+\.[0-9]+\.[0-9]+ each sub
                number < 255
                prefix_length = [0-9]+ and with value <= 32.
                A prefix_length of 24  matches the subnet mask
                255.255.255.0.
                 A prefix_length of 32 requires  matching of the exact IP.
<port range>
                start-port[-end-port] where port numbers are >0 and <65536
```

Note that rules are evaluated in the order of definition. So the first match wins. If no match is made, `libsdp` will default to `both`.

Examples:

- Use SDP by clients connecting to machines that belongs to subnet 192.168.1.*

```
use sdp connect *  192.168.1.0/24:*
```

- Use SDP by ttcp when it connects to port 5001 of any machine

```
use sdp    listen  ttcp      *:5001
```

- Use TCP for any program with name starting with ttcp* serving ports 22 to 25

```
use tcp    server  ttcp*     *:22-25
```

- Listen on both TCP and SDP by any server that listen on port 8080

```
use both   server  *         *:8080
```

- Connect ssh through SDP and fallback to TCP to hosts on 11.4.8.* port 22

```
use both   connect *         11.4.8.0/24:22
```

3.3.2.4.2 Explicit (Non-transparent) Conversion

Use explicit conversion if you need to maintain full control from your application while using SDP. To configure an explicit conversion to use SDP, simply recompile the application replacing PF_INET (or PF_INET) with AF_INET_SDP (or AF_INET_SDP) when calling the `socket()` system call in the source code. The value of AF_INET_SDP is defined in the file `sdp_socket.h` or you can define it inline:

```
#define AF_INET_SDP 27
#define PF_INET_SDP AF_INET_SDP
```

You can compile and execute the following very simple TCP application that has been converted explicitly to SDP:

Compilation:

```
gcc sdp_server.c -o sdp_server
gcc sdp_client.c -o sdp_client
Usage:
Server:
    host1$ sdp_server
Client:
    host1$ sdp_client <server IP addr>
```

Example:

Server:

```
host1$ ./sdp_server
accepted connection from 15.2.2.42:48710
read 2048 bytes
end of test
host1$
```

Client:

```
host2$  ./sdp_client 15.2.2.43
connected to 15.2.2.43:22222
sent 2048 bytes
host2$
```

```
sdp_client.c Code

/*
 *  usage: ./sdp_client <ip_addr>
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define DEF_PORT 22222

#define AF_INET_SDP 27
#define PF_INET_SDP AF_INET_SDP

#define TXBUFSZ 2048
uint8_t tx_buffer[TXBUFSZ];

int
main(int argc, char **argv)
{
    if ( argc < 2) {
        printf("Usage: sdp_client <ip_addr>\n");
        exit(EXIT_FAILURE);
    }

    int sd = socket(PF_INET_SDP, SOCK_STREAM, 0);
    if ( sd < 0) {
        perror("socket() failed");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in to_addr = {
        .sin_family = AF_INET,
        .sin_port = htons(DEF_PORT),
    };

    int ip_ret = inet_aton(argv[1], &to_addr.sin_addr);
    if ( ip_ret == 0) {
        printf("invalid ip address '%s'\n", argv[1]);
        exit(EXIT_FAILURE);
    }

    int conn_ret = connect(sd, (struct sockaddr *) &to_addr, sizeof(to_addr));
    if ( conn_ret < 0) {
        perror("connect() failed");
        exit(EXIT_FAILURE);
    }

    printf("connected to %s:%u\n",
            inet_ntoa(to_addr.sin_addr),
            ntohs(to_addr.sin_port) );


    ssize_t nw = write(sd, tx_buffer, TXBUFSZ);
    if ( nw < 0) {
        perror("write() failed");
        exit(EXIT_FAILURE);
    } else if ( nw == 0) {
        printf("socket was closed by remote host\n");
    }

    printf("sent %zd bytes\n", nw);

    close(sd);

    return 0;
}

sdp_server.c Code

/*
 * Usage: ./sdp_server
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/epoll.h>
```

```
#include <errno.h>
#include <assert.h>

#define RXBUFSZ 2048

uint8_t rx_buffer[RXBUFSZ];

#define DEF_PORT 22222

#define AF_INET_SDP 27
#define PF_INET_SDP AF_INET_SDP

int
main(int argc, char **argv)
{

    int sd = socket(PF_INET_SDP, SOCK_STREAM, 0);
    if ( sd < 0) {
        perror("socket() failed");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in my_addr = {
        .sin_family = AF_INET,
        .sin_port = htons(DEF_PORT),
        .sin_addr.s_addr = INADDR_ANY,
    };


    int retbind = bind(sd, (struct sockaddr *) &my_addr, sizeof(my_addr) );
    if ( retbind < 0) {
        perror("bind() failed");
        exit(EXIT_FAILURE);
    }

    int retlisten = listen(sd, 5/*backlog*/);
    if ( retlisten < 0) {
        perror("listen() failed");
        exit(EXIT_FAILURE);
    }

    // accept the client connection
    struct sockaddr_in client_addr;

    socklen_t client_addr_len = sizeof(client_addr);
    int cd = accept(sd, (struct sockaddr *) &client_addr, &client_addr_len);
    if ( cd < 0) {
        perror("accept() failed");
        exit(EXIT_FAILURE);
    }

    printf("accepted connection from %s:%u\n",
            inet_ntoa(client_addr.sin_addr),
            ntohs(client_addr.sin_port) );

    ssize_t nr = read(cd, rx_buffer, RXBUFSZ);
    if ( nr < 0) {
        perror("read() failed");
        exit(EXIT_FAILURE);
    } else if ( nr == 0) {
        printf("socket was closed by remote host\n");
    }

    printf("read %zd bytes\n", nr);

    printf("end of test\n");

    close(cd);
    close(sd);

    return 0;
}
```

3.3.2.5 BZCopy – Zero Copy Send

BZCOPY mode is only effective for large block transfers. By setting the `/sys` parameter `'sdp_zcopy_thresh'` to a non-zero value, a non-standard SDP speedup is enabled. Messages longer than `'sdp_zcopy_thresh'` bytes in length cause the user space buffer to be pinned and the data to be sent directly from the original buffer. This results in less CPU usage and, on many systems, much higher bandwidth.

Note that the default value of `'sdp_zcopy_thresh'` is 64KB, but is may be too low for some systems. You will need to experiment with your hardware to find the best value.

3.3.2.6 Using RDMA for Small Buffers

For smaller buffers, the overhead of preparing a user buffer to be RDMA'ed is too big; therefore, it is more efficient to use BCopy. (Large buffers can also be sent using RDMA, but they lower CPU utilization.) This mode is called “ZCopy combined mode”. The sendmsg syscall is blocked until the buffer is transfered to the socket's peer, and the data is copied directly from the user buffer at the source side to the user buffer at the sink side.

To set the threshold, use the module parameter sdp_zcopy_thresh. This parameter can be accessed through sysfs (/sys/module/ib_sdp/parameters/sdp_zcopy_thresh). Setting it to 0, disables ZCopy.

3.3.3 iSCSI Extensions for RDMA (iSER)

iSCSI Extensions for RDMA (iSER) extends the iSCSI protocol to RDMA. It permits data to be transferred directly into and out of SCSI buffers without intermediate data copies.

3.3.3.1 iSER Initiator

Setting the iSER target is out of scope of this manual. For guidelines of how to do so, please refer to the relevant target documentation (e.g. stgt, clitarget).

The iSER initiator is controlled through the iSCSI interface available from the iscsi-initiator-utils package.

Make sure iSCSI is enabled and properly configured on your system before proceeding with iSER. Additionally, make sure you have RDMA connectivity between the initiator and the target.

```
rping -s [-vVd] [-S size] [-C count] [-a addr] [-p port]
```

Targets settings such as `timeouts` and `retries` are set the same as any other iSCSI targets.

If targets are set to auto connect on boot, and targets are unreachable, it may take a long time to continue the boot process if `timeouts` and `max retries` are set too high.

Example for discovering and connecting targets over iSER:

```
iscsiadm -m discovery -o new -o old -t st -I iser -p <ip:port> -l
```

iSER also supports RoCE without any additional configuration required. To bond the RoCE interfaces, set the `fail_over_mac` option in the bonding driver (see Section 3.2.5.1.5, “Bonding IPoIB”, on page 146).

For further information, please refer to Mellanox Community (http://community.mellanox.com/community/solutions/content --> iSER).

3.3.4 Lustre

Lustre Compilation for MLNX_OFED:

This procedure applies to RHEL/SLES OSs supported by Lustre. For further information, please refer to Lustre Release Notes.

➢ ***To compile Lustre version 2.4.0 and higher:***

```
$ ./configure --with-o2ib=/usr/src/ofa_kernel/default/
$ make rpms
```

➢ ***To compile older Lustre versions:***

```
$ EXTRA_LNET_INCLUDE="-I/usr/src/ofa_kernel/default/include/ -include /usr/src/
ofa_kernel/default/include/linux/compat-2.6.h" ./configure --with-o2ib=/usr/src/
ofa_kernel/default/
$ EXTRA_LNET_INCLUDE="-I/usr/src/ofa_kernel/default/include/ -include /usr/src/
ofa_kernel/default/include/linux/compat-2.6.h" make rpms
```

3.4 Virtualization

3.4.1 Single Root IO Virtualization (SR-IOV)

Single Root IO Virtualization (SR-IOV) is a technology that allows a physical PCIe device to present itself multiple times through the PCIe bus. This technology enables multiple virtual instances of the device with separate resources. Mellanox adapters are capable of exposing in ConnectX®-3 adapter cards up to 126 virtual instances called Virtual Functions (VFs) and ConnectX-4/Connect-IB adapter cards up to 62 virtual instances. These virtual functions can then be provisioned separately. Each VF can be seen as an additional device connected to the Physical Function. It shares the same resources with the Physical Function, and its number of ports equals those of the Physical Function.

SR-IOV is commonly used in conjunction with an SR-IOV enabled hypervisor to provide virtual machines direct hardware access to network resources hence increasing its performance.

In this chapter we will demonstrate setup and configuration of SR-IOV in a Red Hat Linux environment using Mellanox ConnectX® VPI adapter cards family.

3.4.1.1 System Requirements

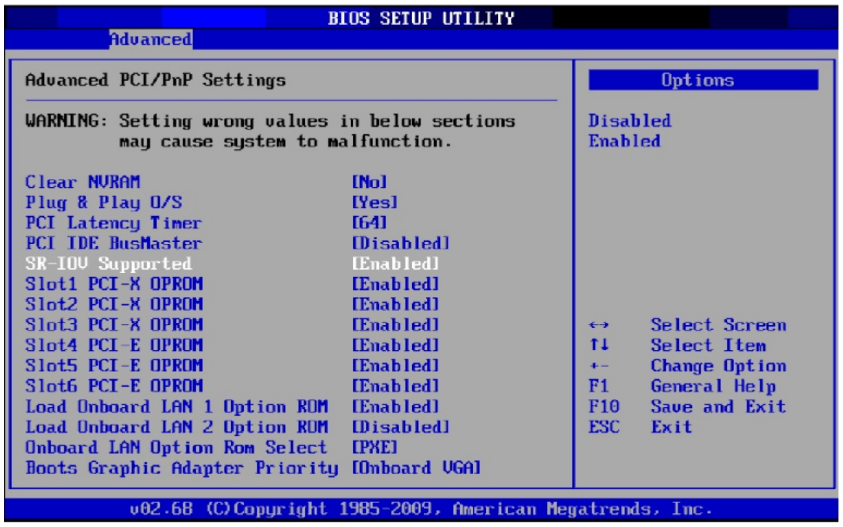
To set up an SR-IOV environment, the following is required:

- MLNX_OFED Driver
- A server/blade with an SR-IOV-capable motherboard BIOS
- Hypervisor that supports SR-IOV such as: Red Hat Enterprise Linux Server Version 6.*
- Mellanox ConnectX® VPI Adapter Card family with SR-IOV capability

3.4.1.2 Setting Up SR-IOV

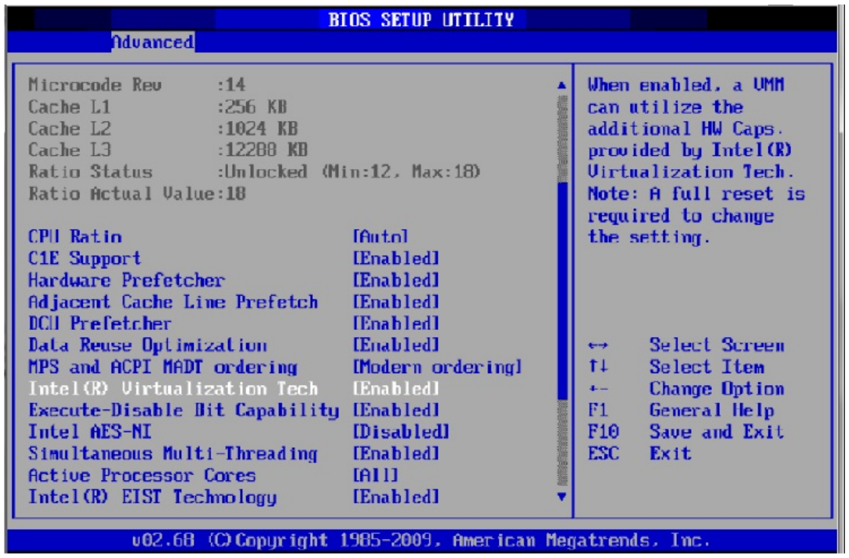
Depending on your system, perform the steps below to set up your BIOS. The figures used in this section are for illustration purposes only. For further information, please refer to the appropriate BIOS User Manual:

Step 1. Enable "SR-IOV" in the system BIOS.



```
Advanced PCI/PnP Settings

WARNING: Setting wrong values in below sections
         may cause system to malfunction.

Clear NVRAM                        [No]
Plug & Play O/S                    [Yes]
PCI Latency Timer                  [64]
PCI IDE BusMaster                  [Disabled]
SR-IOV Supported                   [Enabled]
Slot1 PCI-X OPROM                  [Enabled]
Slot2 PCI-X OPROM                  [Enabled]
Slot3 PCI-X OPROM                  [Enabled]
Slot4 PCI-E OPROM                  [Enabled]
Slot5 PCI-E OPROM                  [Enabled]
Slot6 PCI-E OPROM                  [Enabled]
Load Onboard LAN 1 Option ROM      [Enabled]
Load Onboard LAN 2 Option ROM      [Disabled]
Onboard LAN Option Rom Select      [PXE]
Boots Graphic Adapter Priority     [Onboard VGA]

Disabled
Enabled

↔    Select Screen
↑↓   Select Item
+-   Change Option
F1   General Help
F10  Save and Exit
ESC  Exit

v02.68 (C)Copyright 1985-2009, American Megatrends, Inc.
```

Step 2. Enable “Intel Virtualization Technology”.



```
Microcode Rev       :14
Cache L1            :256 KB
Cache L2            :1024 KB
Cache L3            :12288 KB
Ratio Status        :Unlocked (Min:12, Max:18)
Ratio Actual Value:18

CPU Ratio                          [Auto]
C1E Support                        [Enabled]
Hardware Prefetcher                [Enabled]
Adjacent Cache Line Prefetch       [Enabled]
DCU Prefetcher                     [Enabled]
Data Reuse Optimization            [Enabled]
MPS and ACPI MADT ordering         [Modern ordering]
Intel(R) Virtualization Tech       [Enabled]
Execute-Disable Bit Capability     [Enabled]
Intel AES-NI                       [Disabled]
Simultaneous Multi-Threading       [Enabled]
Active Processor Cores             [All]
Intel(R) EIST Technology           [Enabled]

When enabled, a VMM
can utilize the
additional HW Caps.
provided by Intel(R)
Virtualization Tech.
Note: A full reset is
required to change
the setting.

↔    Select Screen
↑↓   Select Item
+-   Change Option
F1   General Help
F10  Save and Exit
ESC  Exit

v02.68 (C)Copyright 1985-2009, American Megatrends, Inc.
```

Step 3. Install a hypervisor that supports SR-IOV.

Step 4. Depending on your system, update the /boot/grub/grub.conf file to include a similar command line load parameter for the Linux kernel.

For example, to Intel systems, add:

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.32-36.x86-645)
        root (hd0,0)
        kernel /vmlinuz-2.6.32-36.x86-64 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
        intel_iommu=on[1]
        initrd /initrd-2.6.32-36.x86-64.img
```

1. Please make sure the parameter "intel_iommu=on" exists when updating the /boot/grub/grub.conf file, otherwise SR-IOV cannot be loaded.

3.4.1.2.1 Configuring SR-IOV for ConnectX-3/ConnectX-3 Pro

Step 1. Install the MLNX_OFED driver for Linux that supports SR-IOV.

SR-IOV can be enabled and managed by using one of the following methods:

- Run the mlxconfig tool and set the `SRIOV_EN` parameter to "`1`" without re-burning the firmware

  To find the mst device run: "`mst start`" and "`mst status`"

  ```
  mlxconfig -d <mst_device> s SRIOV_EN=1
  ```

  For further information, please refer to section *"mlxconfig - Changing Device Configuration Tool"* in the MFT User Manual (www.mellanox.com > Products > Software > Firmware Tools).

- Burn firmware with SR-IOV support where the number of virtual functions (VFs) will be set to 16

  ```
  --enable-sriov
  ```

Step 2. Verify the HCA is configured to support SR-IOV.

```
# mstflint -dev <PCI Device> dc
```

1. Verify in the [HCA] section the following fields appear[1],[2]:

```
[HCA]
num_pfs = 1
total_vfs = <0-126>
sriov_en = true
```

Parameter	Recommended Value
num_pfs	1 **Note:** This field is optional and might not always appear.

1. If SR-IOV is supported, to enable SR-IOV (if it is not enabled), it is sufficient to set "sriov_en = true" in the INI.
2. If the HCA does not support SR-IOV, please contact Mellanox Support: support@mellanox.com

Parameter	Recommended Value
total_vfs	• When using firmware version 2.31.5000 and above, the recommended value is 126. • When using firmware version 2.30.8000 and below, the recommended value is 63 **Note:** Before setting number of VFs in SR-IOV, please make sure your system can support that amount of VFs. Setting number of VFs larger than what your Hardware and Software can support may cause your system to cease working.
sriov_en	true

2. Add the above fields to the INI if they are missing.
3. Set the `total_vfs` parameter to the desired number if you need to change the number of total VFs.
4. Reburn the firmware using the mlxburn tool if the fields above were added to the INI, or the `total_vfs` parameter was modified.
   If the mlxburn is not installed, please downloaded it from the Mellanox website http://www.mellanox.com > products > Firmware tools

```
mlxburn -fw ./fw-ConnectX3-rel.mlx -dev /dev/mst/mt4099_pci_cr0 -conf ./MCX341A-
XCG_Ax.ini
```

Step 3. Create the text file /etc/modprobe.d/mlx4_core.conf if it does not exist.

Step 4. Insert an "option" line in the /etc/modprobe.d/mlx4_core.conf file to set the number of VFs. the protocol type per port, and the allowed number of virtual functions to be used by the physical function driver (probe_vf).

For example:

```
options mlx4_core num_vfs=5 port_type_array=1,2 probe_vf=1
```

Parameter	Recommended Value
num_vfs	• If absent, or zero: no VFs will be available • If its value is a single number in the range of 0-63: The driver will enable the `num_vfs` VFs on the HCA and this will be applied to all ConnectX® HCAs on the host. • If its a triplet x,y,z (applies only if all ports are configured as Ethernet) the driver creates: • x single port VFs on physical port 1 • y single port VFs on physical port 2 (applies only if such a port exist) • z n-port VFs (where n is the number of physical ports on device). This applies to all ConnectX® HCAs on the host

Parameter	Recommended Value
num_vfs	• If its format is a string: The string specifies the `num_vfs` parameter separately per installed HCA. The string format is: "bb:dd.f-v,bb:dd.f-v,…" • bb:dd.f = bus:device.function of the PF of the HCA • v = number of VFs to enable for that HCA which is either a single value or a triplet, as described above. For example: • `num_vfs=5` - The driver will enable 5 VFs on the HCA and this will be applied to all ConnectX® HCAs on the host • `num_vfs=00:04.0-5,00:07.0-8` - The driver will enable 5 VFs on the HCA positioned in BDF 00:04.0 and 8 on the one in 00:07.0) • `num_vfs=1,2,3` - The driver will enable 1 VF on physical port 1, 2 VFs on physical port 2 and 3 dual port VFs (applies only to dual port HCA when all ports are Ethernet ports). • `num_vfs=00:04.0-5;6;7,00:07.0-8;9;10` - The driver will enable: • HCA positioned in BDF 00:04.0 • 5 single VFs on port 1 • 6 single VFs on port 2 • 7 dual port VFs • HCA positioned in BDF 00:07.0 • 8 single VFs on port 1 • 9 single VFs on port 2 • 10 dual port VFs Applies when all ports are configure as Ethernet in dual port HCAs **Notes:** • PFs not included in the above list will not have SR-IOV enabled. • Triplets and single port VFs are only valid when all ports are configured as Ethernet. When an InfiniBand port exists, only `num_vfs=a` syntax is valid where `"a"` is a single value that represents the number of VFs. • The second parameter in a triplet is valid only when there are more than 1 physical port. In a triplet, `x+z<=63` and `y+z<=63`, the maximum number of VFs on each physical port must be 63.
port_type_array	Specifies the protocol type of the ports. It is either one array of 2 port types `'t1,t2'` for all devices or list of BDF to `port_type_array 'bb:dd.f-t1;t2,...'`. (string) Valid port types: 1-ib, 2-eth, 3-auto, 4-N/A If only a single port is available, use the N/A port type for port2 (e.g '1,4'). Note that this parameter is valid only when `num_vfs` is not zero (i.e., SRIOV is enabled). Otherwise, it is ignored.

Parameter	Recommended Value
probe_vf	• If absent or zero: no VF interfaces will be loaded in the Hypervisor/host • If num_vfs is a number in the range of 1-63, the driver running on the Hypervisor will itself activate that number of VFs. All these VFs will run on the Hypervisor. This number will apply to all ConnectX® HCAs on that host. • If its a triplet x,y,z (applies only if all ports are configured as Ethernet), the driver probes: • x single port VFs on physical port 1 • y single port VFs on physical port 2 (applies only if such a port exist) • z n-port VFs (where n is the number of physical ports on device). Those VFs are attached to the hypervisor. • If its format is a string: the string specifies the `probe_vf` parameter separately per installed HCA. The string format is: "bb:dd.f-v,bb:dd.f-v,… • bb:dd.f = bus:device.function of the PF of the HCA • v = number of VFs to use in the PF driver for that HCA which is either a single value or a triplet, as described above For example: • `probe_vfs=5` - The PF driver will activate 5 VFs on the HCA and this will be applied to all ConnectX® HCAs on the host • `probe_vfs=00:04.0-5,00:07.0-8` - The PF driver will activate 5 VFs on the HCA positioned in BDF 00:04.0 and 8 for the one in 00:07.0) • `probe_vf=1,2,3` - The PF driver will activate 1 VF on physical port 1, 2 VFs on physical port 2 and 3 dual port VFs (applies only to dual port HCA when all ports are Ethernet ports). This applies to all ConnectX® HCAs in the host. • `probe_vf=00:04.0-5;6;7,00:07.0-8;9;10` - The PF driver will activate: • HCA positioned in BDF 00:04.0 • 5 single VFs on port 1 • 6 single VFs on port 2 • 7 dual port VFs • HCA positioned in BDF 00:07.0 • 8 single VFs on port 1 • 9 single VFs on port 2 • 10 dual port VFs Applies when all ports are configure as Ethernet in dual port HCAs.

Parameter	Recommended Value
probe_vf	**Notes:** • PFs not included in the above list will not activate any of their VFs in the PF driver • Triplets and single port VFs are only valid when all ports are configured as Ethernet. When an InfiniBand port exist, only `probe_vf=a` syntax is valid where "`a`" is a single value that represents the number of VFs • The second parameter in a triplet is valid only when there are more than 1 physical port • Every value (either a value in a triplet or a single value) should be less than or equal to the respective value of `num_vfs` parameter

The example above loads the driver with 5 VFs (num_vfs). The standard use of a VF is a single VF per a single VM. However, the number of VFs varies upon the working mode requirements.

The protocol types are:

- Port 1 = IB
- Port 2 = Ethernet
  - port_type_array=2,2 (Ethernet, Ethernet)
  - port_type_array=1,1 (IB, IB)
  - port_type_array=1,2 (VPI: IB, Ethernet)
  - NO port_type_array module parameter: ports are IB

For single port HCAs the possible values are (1,1) or (2,2).

Step 5. Reboot the server.

If the SR-IOV is not supported by the server, the machine might not come out of boot/ load.

Step 6. Load the driver and verify the SR-IOV is supported. Run:

```
lspci | grep Mellanox
03:00.0 InfiniBand: Mellanox Technologies MT26428 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR /
10GigE] (rev b0)
03:00.1 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function] (rev b0)
03:00.2 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function] (rev b0)
03:00.3 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function] (rev b0)
03:00.4 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function] (rev b0)
03:00.5 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function] (rev b0)
```

Where:

- "03:00" represents the Physical Function
- "03:00.**X**" represents the Virtual Function connected to the Physical Function

3.4.1.2.2 Configuring SR-IOV for ConnectX-4/Connect-IB

Step 1. Install the MLNX_OFED driver for Linux that supports SR-IOV.

Step 2. Check if SR-IOV is enabled in the firmware.

```
mlxconfig -d /dev/mst/mt4113_pciconf0 q

  Device #1:
  ----------


  Device type:    ConnectIB
  PCI device:     /dev/mst/mt4113_pciconf0
  Configurations:          Current
     SRIOV_EN              1
     NUM_OF_VFS            8
     FPP_EN                1
```

If not, use mlxconfig to enable it.

```
mlxconfig -d /dev/mst/mt4113_pciconf0 set SRIOV_EN=1 NUM_OF_VFS=16 FPP_EN=1
```

Step 3. Either reset or reboot the firmware.

```
mlxfwreset / reboot
```

Step 4. Write to the sysfs file the number of Virtual Functions you need to create for the PF.

You can use one of the following files:

- A standard Linux kernel generated file that is available in the new kernels.

  ```
  echo [num_vfs] > /sys/class/infiniband/mlx5_0/device/sriov_numvfs
  ```

- A file generated by the mlx5_core driver with the same functionality as the kernel generated one. Used by old kernels that do not have the standard file.

  ```
  echo [num_vfs] > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
  ```

The following rules apply when writing to these file:

- If there are no VFs assigned, the number of VFs can be changed to any valid value (0 - max #VFs as set during FW burning)
- If there are VFs assigned to a VM, it is not possible to change the number of VFs
- If the administrator unloads the driver on the PF while there are no VFs assigned, the driver will unload and SRI-OV will be disabled
- If there are VFs assigned while the driver of the PF is unloaded, SR-IOV is not be disabled. This means VFs will be visible on the VM. However they will not be operational.
  - The VF driver will discover this situation and will close its resources
  - When the driver on the PF is reloaded, the VF becomes operational. The administrator of the VF will need to restart the driver in order to resume working with the VF.

Step 5. Load the driver and verify the SR-IOV is supported. Run:

```
lspci | grep Mellanox
08:00.0 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4]
08:00.1 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4]
08:00.2 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual
Function]
08:00.3 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual
Function]
08:00.4 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual
Function]
08:00.5 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual
Function]
```

When SR-IOV is enabled in the PF, the following structure becomes available in at /sys/class/infiniband/mlx5_0/device/sriov

```
+-- 0
|   +-- node
|   +-- policy
|   +-- port
+-- 1
|   +-- node
|   +-- policy
|   +-- port
+-- 2
    +-- node
    +-- policy
    +-- port
```

In the extract above we see three Virtual Functions numbered 0 to 2. For each Virtual Function we have the following files:

- Node stands for node GUID: The user can set the node GUID of the VF by writing to this file.
- Port stands for port GUID: The user can set the port GUID of the VF by writing to this file.
- Policy: The user can read or modify the vport policy.
  - Policy can be one of:
    - Down - the VPort PortState always remains 'Down'
    - Up - if the current VPort PortState is 'Down', it is modified to 'Initialize'. In all other states, it is unmodified. The result is that the SM may bring the VPort up.
    - Follow - follows the PortState of the physical port. If the PortState of the physical port is 'Active', then the VPort implements the 'Up' policy. Otherwise, the VPort PortState is 'Down'.
  - The policy of all the vports is initialized to Down after restart of the PF driver.
    - An exception is vport0 which for which the policy is modified to Follow by the PF driver

3.4.1.2.3 Note on VFs Initialization

Since the same `mlx5_core` driver supports both Physical and Virtual Functions, once the Virtual Functions are created, the driver of the PF will attempt to initialize them so they will be available to the OS owning the PF. If you want to assign a Virtual Function to a VM, you need to make sure the VF is not used by the PF driver. If a VF is used, you should first unbind it before assigning to a VM.

➢ ***To unbind a device use the following command:***

1. Get the full PCI address of the device.

```
lspci -D
```

Example:

```
0000:09:00.2
```

2. Unbind the device.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Bind the unbound VF.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

3.4.1.3 Assigning a Virtual Function to a Virtual Machine

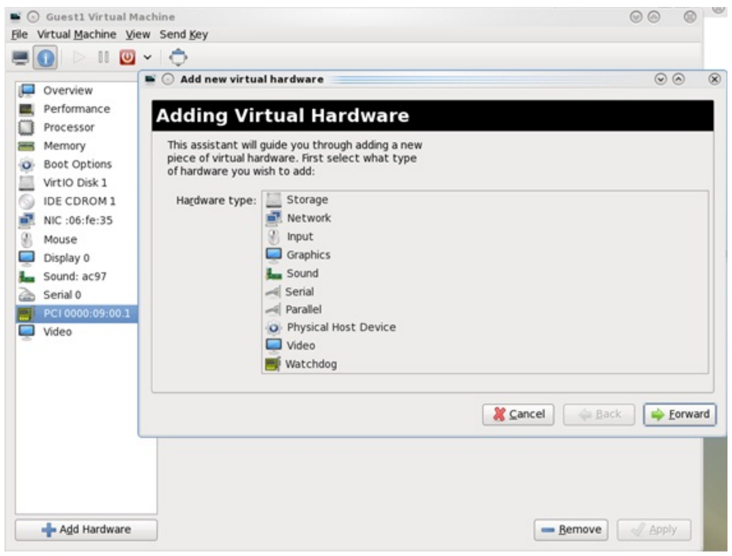
This section will describe a mechanism for adding a SR-IOV VF to a Virtual Machine.

Assigning the SR-IOV Virtual Function to the Red Hat KVM VM Server

Step 1. Run the virt-manager.

Step 2. Double click on the virtual machine and open its Properties.

Step 3. Go to Details->Add hardware ->PCI host device.



This assistant will guide you through adding a new piece of virtual hardware. First select what type of hardware you wish to add:

Step 4. Choose a Mellanox virtual function according to its PCI device (e.g., 00:03.1)

Step 5. If the Virtual Machine is up reboot it, otherwise start it.

Step 6. Log into the virtual machine and verify that it recognizes the Mellanox card. Run:

```
lspci | grep Mellanox

00:03.0 InfiniBand: Mellanox Technologies MT27500 Family [ConnectX-3 Virtual Function]
(rev b0)
```

Step 7. Add the device to the `/etc/sysconfig/network-scripts/ifcfg-ethX` configuration file. The MAC address for every virtual function is configured randomly, therefore it is not necessary to add it.

3.4.1.4 Uninstalling SR-IOV Driver

➢ ***To uninstall SR-IOV driver, perform the following:***

Step 1. For Hypervisors, detach all the Virtual Functions (VF) from all the Virtual Machines (VM) or stop the Virtual Machines that use the Virtual Functions.

Please be aware, stopping the driver when there are VMs that use the VFs, will cause machine to hang.

Step 2. Run the script below. Please be aware, uninstalling the driver deletes the entire driver's file, but does not unload the driver.

```
[root@swl022 ~]# /usr/sbin/ofed_uninstall.sh
This program will uninstall all OFED packages on your machine.
Do you want to continue?[y/N]:y
Running /usr/sbin/vendor_pre_uninstall.sh
Removing OFED Software installations
Running /bin/rpm -e --allmatches kernel-ib kernel-ib-devel libibverbs libibverbs-devel
libibverbs-devel-static libibverbs-utils libmlx4 libmlx4-devel libibcm libibcm-devel
libibumad libibumad-devel libibumad-static libibmad libibmad-devel libibmad-static
librdmacm librdmacm-utils librdmacm-devel ibacm opensm-libs opensm-devel perftest com-
pat-dapl compat-dapl-devel dapl dapl-devel dapl-devel-static dapl-utils srptools infini-
band-diags-guest ofed-scripts opensm-devel
warning: /etc/infiniband/openib.conf saved as /etc/infiniband/openib.conf.rpmsave
Running /tmp/2818-ofed_vendor_post_uninstall.sh
```

Step 3. Restart the server.

3.4.1.5 Configuring Pkeys and GUIDs under SR-IOV

Port Type Management

Port Type management is static when enabling SR-IOV (the `connectx_port_config` script will not work). The port type is set on the Host via a module parameter, `port_type_array`, in `mlx-4_core`. This parameter may be used to set the port type uniformly for all installed ConnectX® HCAs, or it may specify an individual configuration for each HCA.

This parameter should be specified as an options line in the file `/etc/modprobe.d/mlx-4_core.conf`.

For example, to configure all HCAs to have Port1 as IB and Port2 as ETH, insert the following line:

```
options mlx4_core port_type_array=1,2
```

To set HCAs individually, you may use a string of `Domain:bus:device.function=x;y`

For example, if you have a pair of HCAs, whose PFs are 0000:04:00.0 and 0000:05:00.0, you may specify that the first will have both ports as IB, and the second will have both ports as ETH as follows:

```
options mlx4_core port_type_array='0000:04:00.0-1;1,0000:05:00.0-2;2
```

Only the PFs are set via this mechanism. The VFs inherit their port types from their associated PF.

Virtual Function InfiniBand Ports

Each VF presents itself as an independent vHCA to the host, while a single HCA is observable by the network which is unaware of the vHCAs. No changes are required by the InfiniBand subsystem, ULPs, and applications to support SR-IOV, and vHCAs are interoperable with any existing (non-virtualized) IB deployments.

Sharing the same physical port(s) among multiple vHCAs is achieved as follows:

- Each vHCA port presents its own virtual GID table

  For further details, please refer to Section 3.4.1.5.3, on page 210.
- Each vHCA port presents its own virtual PKey table

  The virtual PKey table (presented to a VF) is a mapping of selected indexes of the physical PKey table. The host admin can control which PKey indexes are mapped to which virtual indexes using a sysfs interface. The physical PKey table may contain both full and partial memberships of the same PKey to allow different membership types in different virtual tables.
- Each vHCA port has its own virtual port state

  A vHCA port is up if the following conditions apply:
  - The physical port is up
  - The virtual GID table contains the GIDs requested by the host admin
  - The SM has acknowledged the requested GIDs since the last time that the physical port went up
- Other port attributes are shared, such as: GID prefix, LID, SM LID, LMC mask

To allow the host admin to control the virtual GID and PKey tables of vHCAs, a new sysfs 'iov sub-tree has been added under the PF InfiniBand device.

If the vHCA comes up without a GUID, make sure you are running the latest version of SM/OpenSM. The SM on QDR switches do not support SR-IOV.

3.4.1.5.1 SR-IOV sysfs Administration Interfaces on the Hypervisor

Administration of GUIDs and PKeys is done via the sysfs interface in the Hypervisor (Dom0). This interface is under:

```
/sys/class/infiniband/<infiniband device>/iov
```

Under this directory, the following subdirectories can be found:

- `ports` - The actual (physical) port resource tables

  Port GID tables:
  - `ports/<n>/gids/<n> where 0 <= n <= 127` (the physical port gids)
  - `ports/<n>/admin_guids/<n> where 0 <= n <= 127` (allows examining or changing the administrative state of a given GUID>
  - `ports/<n>/pkeys/<n> where 0 <= n <= 126` (displays the contents of the physical pkey table)
- `<pci id> directories` - one for Dom0 and one per guest. Here, you may see the mapping between virtual and physical pkey indices, and the virtual to physical gid 0.

Currently, the GID mapping cannot be modified, but the pkey virtual to physical mapping can .

These directories have the structure:

- `<pci_id>/port/<m>/gid_idx/0 where m = 1..2` (this is read-only)

  and
- `<pci_id>/port/<m>/pkey_idx/<n>`, where `m = 1..2` and `n = 0..126`

For instructions on configuring pkey_idx, please see below.

3.4.1.5.2 Configuring an Alias GUID (under ports/<n>/admin_guids)

Step 1. Determine the GUID index of the PCI Virtual Function that you want to pass through to a guest.

For example, if you want to pass through PCI function 02:00.3 to a certain guest, you initially need to see which GUID index is used for this function.

To do so:

```
cat /sys/class/infiniband/iov/0000:02:00.3/port/<port_num>/gid_idx/0
```

The value returned will present which guid index to modify on Dom0.

Step 2. Modify the physical GUID table via the `admin_guids` sysfs interface.

To configure the GUID at index `<n>` on port `<port_num>`:

```
cd /sys/class/infiniband/mlx4_0/iov/ports/<port_num>/admin_guids
echo <your desired guid>  > n
```

Example:

```
cd /sys/class/infiniband/mlx4_0/iov/ports/1/admin_guids
echo1 "0x002fffff8118" > 3
```

1. echo "0x0" means let the SM assign a value to that GUID
   echo "0xffffffffffffffff" means delete that GUID
   echo <any other value> means request the SM to assign this GUID to this index

Step 3. Read the administrative status of the GUID index.

To read the administrative status of GUID index `m` on port `n`:

```
cat /sys/class/infiniband/mlx4_0/iov/ports/<n>/admin_guids/<m>
```

Step 4. Check the operational state of a GUID.

```
 /sys/class/infiniband/mlx4_0/iov/ports/<n>/gids  (where n = 1 or 2)
```

The values indicate what gids are actually configured on the firmware/hardware, and all the entries are R/O.

Step 5. Compare the value you read under the `"admin_guids"` directory at that index with the value under the `"gids"` directory, to verify the change requested in Step 3 has been accepted by the SM, and programmed into the hardware port GID table.

If the value under `admin_guids/<m>` is different that the value under `gids/<m>`, the request is still in progress.

3.4.1.5.3 Alias GUID Support in InfiniBand

3.4.1.5.3.1 Admin VF GUIDs

As of MLNX_OFED v3.0, the `query_gid` verb (e.g. `ib_query_gid()`) returns the admin desired value instead of the value that was approved by the SM to prevent a case where the SM is unreachable or a response is delayed, or if the VF is probed into a VM before their GUID is registered with the SM. If one of the above scenarios occurs, the VF sees an incorrect GID (i.e., not the GID that was intended by the admin).

Despite the new behavior, if the SM does not approve the GID, the VF sees its link as down.

3.4.1.5.3.2 On Demand GUIDs

GIDs are requested from the SM on demand, when needed by the VF (e.g. become active), and are released when the GIDs are no longer in use.

Since a GID is assigned to a VF on the destination HCA, while the VF on the source HCA is shut down (but not administratively released), using GIDs on demand eases the GID migrations.

For compatibility reasons, an explicit admin request to set/change a GUID entry is done immediately, regardless of whether the VF is active or not to allow administrators to change the GUID without the need to unbind/bind the VF.

3.4.1.5.3.3 Alias GUIDs Default Mode

Due to the change in the Alias GUID support in InfiniBand behavior, its default mode is now set as HOST assigned instead of SM assigned. To enable out-of-the-box experience, the PF generates random GUIDs as the initial admin values instead of asking the SM.

3.4.1.5.3.4 Initial GUIDs' Values

Initial GUIDs' values depend on the `mlx4_ib` module parameter `'sm_guid_assign'` as follows:

Mode Type	Description
admin assigned	Each admin_guid entry has the random generated GUID value.
sm assigned	Each admin_guid entry for non-active VFs has a value of 0. Meaning, asking a GUID from the SM upon VF activation. When a VF is active, the returned value from the SM becomes the admin value to be asked later again.

When a VF becomes active, and its admin value is approved, the operational GID entry is changed accordingly. In both modes, the administrator can set/delete the value by using the sysfs Administration Interfaces on the Hypervisor as described above.

3.4.1.5.3.5 Single GUID per VF

Each VF has a single GUID entry in the table based on the VF number. (e.g. VF 1 expects to use GID entry 1). To determine the GUID index of the PCI Virtual Function to pass to a guest, use the sysfs mechanism <gid_idx> directory as described above.

3.4.1.5.3.6 Persistency Support

Once admin request is rejected by the SM, a retry mechanism is set. Retry time is set to 1 second, and for each retry it is multiplied by 2 until reaching the maximum value of 60 seconds. Additionally, when looking for the next record to be updated, the record having the lowest time to be executed is chosen.

Any value reset via the `admin_guid` interface is immediately executed and it resets the entry’s timer.

3.4.1.5.4 Partitioning IPoIB Communication using PKeys

PKeys are used to partition IPoIB communication between the Virtual Machines and the Dom0 by mapping a non-default full-membership PKey to virtual index 0, and mapping the default PKey to a virtual pkey index other than zero.

The below describes how to set up two hosts, each with 2 Virtual Machines. Host-1/vm-1 will be able to communicate via IPoIB only with Host2/vm1,and Host1/vm2 only with Host2/vm2.

In addition, Host1/Dom0 will be able to communicate only with Host2/Dom0 over ib0. vm1 and vm2 will not be able to communicate with each other, nor with Dom0.

This is done by configuring the virtual-to-physical PKey mappings for all the VMs, such that at virtual PKey index 0, both vm-1s will have the same pkey and both vm-2s will have the same PKey (different from the vm-1's), and the Dom0's will have the default pkey (different from the vm's pkeys at index 0).

OpenSM must be used to configure the physical Pkey tables on both hosts.

- The physical Pkey table on both hosts (Dom0) will be configured by OpenSM to be:

```
index 0 = 0xffff
index 1 = 0xb000
index 2 = 0xb030
```

- The vm1's virt-to-physical PKey mapping will be:

```
pkey_idx 0 = 1
pkey_idx 1 = 0
```

- The vm2's virt-to-phys pkey mapping will be:

```
pkey_idx 0 = 2
pkey_idx 1 = 0
```

  so that the default pkey will reside on the vms at index 1 instead of at index 0.

The IPoIB QPs are created to use the PKey at index 0. As a result, the Dom0, vm1 and vm2 IPoIB QPs will all use different PKeys.

➢ ***To partition IPoIB communication using PKeys:***

Step 1. Create a file "`/etc/opensm/partitions.conf`" on the host on which OpenSM runs, containing lines.

```
Default=0x7fff,ipoib : ALL=full ;
Pkey1=0x3000,ipoib : ALL=full;
Pkey3=0x3030,ipoib : ALL=full;
```

This will cause OpenSM to configure the physical Port Pkey tables on all physical ports on the network as follows:

```
pkey idx | pkey value
---------|---------
       0 | 0xFFFF
       1 | 0xB000
       2 | 0xB030
```

(the most significant bit indicates if a PKey is a full PKey).

The "`,ipoib`" causes OpenSM to pre-create IPoIB the broadcast group for the indicated PKeys.

Step 2. Configure (on Dom0) the virtual-to-physical PKey mappings for the VMs.

Step a. Check the PCI ID for the Physical Function and the Virtual Functions.

```
lspci | grep Mel
```

Step b. Assuming that on Host1, the physical function displayed by lspci is "0000:02:00.0", and that on Host2 it is "0000:03:00.0"
On Host1 do the following.

```
cd /sys/class/infiniband/mlx4_0/iov
0000:02:00.0  0000:02:00.1 0000:02:00.2 ...1
```

1. 0000:02:00.0 contains the virtual-to-physical mapping tables for the physical function.
0000:02:00.X contain the virt-to-phys mapping tables for the virtual functions.

Do not touch the Dom0 mapping table (under <nnnn>:<nn>:00.0). Modify only tables under 0000:02:00.1 and/or 0000:02:00.2. We assume that vm1 uses VF 0000:02:00.1 and vm2 uses VF 0000:02:00.2

Step c. Configure the virtual-to-physical PKey mapping for the VMs.

```
echo 0 > 0000:02:00.1/ports/1/pkey_idx/1
echo 1 > 0000:02:00.1/ports/1/pkey_idx/0
echo 0 > 0000:02:00.2/ports/1/pkey_idx/1
echo 2 > 0000:02:00.2/ports/1/pkey_idx/0
```

vm1 pkey index 0 will be mapped to physical pkey-index 1, and vm2 pkey index 0 will be mapped to physical pkey index 2. Both vm1 and vm2 will have their pkey index 1 mapped to the default pkey.

Step d. On Host2 do the following.

```
cd /sys/class/infiniband/mlx4_0/iov
echo 0 > 0000:03:00.1/ports/1/pkey_idx/1
echo 1 > 0000:03:00.1/ports/1/pkey_idx/0
echo 0 > 0000:03:00.2/ports/1/pkey_idx/1
echo 2 > 0000:03:00.2/ports/1/pkey_idx/0
```

Step e. Once the VMs are running, you can check the VM's virtualized PKey table by doing (on the vm).

```
cat /sys/class/infiniband/mlx4_0/ports/[1,2]/pkeys/[0,1]
```

Step 3. Start up the VMs (and bind VFs to them).

Step 4. Configure IP addresses for ib0 on the host and on the guests.

3.4.1.6 Ethernet Virtual Function Configuration when Running SR-IOV

3.4.1.6.1 VLAN Guest Tagging (VGT) and VLAN Switch Tagging (VST)

When running ETH ports on VGT, the ports may be configured to simply pass through packets as is from VFs (Vlan Guest Tagging), or the administrator may configure the Hypervisor to silently force packets to be associated with a VLan/Qos (Vlan Switch Tagging).

In the latter case, untagged or priority-tagged outgoing packets from the guest will have the VLAN tag inserted, and incoming packets will have the VLAN tag removed. Any vlan-tagged packets sent by the VF are silently dropped. The default behavior is VGT.

The feature may be controlled on the Hypervisor from userspace via iprout2 / netlink:

```
ip link set { dev DEVICE | group DEVGROUP } [ { up | down } ]
   ...
   [ vf NUM [ mac LLADDR ]
      [ vlan VLANID [ qos VLAN-QOS ] ]
          ...
   [ spoofchk { on | off} ] ]
          ...
```

use:

```
ip link set dev <PF device> vf <NUM> vlan <vlan_id> [qos <qos>]
```

- where `NUM = 0..max-vf-num`
- vlan_id = `0..4095` (4095 means `"set VGT"`)
- qos = `0..7`

For example:

- `ip link set dev eth2 vf 2 qos 3` - sets VST mode for VF #2 belonging to PF eth2, with qos = 3
- `ip link set dev eth2 vf 2 4095` - sets mode for VF 2 back to VGT

3.4.1.6.2 Additional Ethernet VF Configuration Options

- Guest MAC configuration

  By default, guest MAC addresses are configured to be all zeroes. If the administrator wishes the guest to always start up with the same MAC, he/she should configure guest MACs before the guest driver comes up.

  The guest MAC may be configured by using:

  ```
  ip link set dev <PF device> vf <NUM> mac <LLADDR>
  ```

  For legacy guests, which do not generate random MACs, the adminstrator should always configure their MAC addresses via ip link, as above.

- Spoof checking

  Spoof checking is currently available only on upstream kernels newer than 3.1.

  ```
  ip link set dev <PF device> vf <NUM> spoofchk [on | off]
  ```

3.4.1.6.3 Mapping VFs to Ports

➢ ***To view the VFs mapping to ports:***

Using the ip link tool v2.6.34~3 and above.

```
ip link
```

The output is as following:

```
61: p1p1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group
default qlen 1000
    link/ether 00:02:c9:f1:72:e0 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state auto
    vf 37 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state auto
    vf 38 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off, link-state disable
    vf 39 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off, link-state disable
```

When a MAC is ff:ff:ff:ff:ff:ff, the VF is not assigned to the port of the net device it is listed under. In the example above, **vf 38** is not assigned to the same port as **p1p1**, in contrast to **vf0**.

However, even VFs that are not assigned to the net device, could be used to set and change its settings. For example, the following is a valid command to change the spoof check:

```
ip link set dev p1p1 vf 38 spoofchk on
```

This command will affect only the **vf 38**. The changes can be seen in ip link on the net device that this device is assigned to.

3.4.1.6.4 Mapping VFs to Ports using the mlnx_get_vfs.pl tool

➢ ***To map the PCI representation in BDF to the respective ports:***

```
mlnx_get_vfs.pl
```

The output is as following:

```
BDF 0000:04:00.0
        Port 1: 2
                vf0     0000:04:00.1
                vf1     0000:04:00.2
        Port 2: 2
                vf2     0000:04:00.3
                vf3     0000:04:00.4
        Both: 1
                vf4   0000:04:00.5
```

3.4.1.6.5 RoCE Support

RoCE is supported on Virtual Functions and VLANs may be used with it. For RoCE, the hypervisor GID table size is of 16 entries while the VFs share the remaining 112 entries. When the number of VFs is larger than 56 entries, some of them will have GID table with only a single entry which is inadequate if VF's Ethernet device is assigned with an IP address.

When setting num_vfs in mlx4_core module parameter it is important to check that the number of the assigned IP addresses per VF does not exceed the limit for GID table size.

3.4.1.7 Running Network Diagnostic Tools on a Virtual Function

Until now, in MLNX_OFED, administrators were unable to run network diagnostics from a VF since sending and receiving Subnet Management Packets (SMPs) from a VF was not allowed, for security reasons: SMPs are not restricted by network partitioning and may affect the physical network topology. Moreover, even the SM may be denied access from portions of the network by setting management keys unknown to the SM.

However, it is desirable to grant SMP capability to certain privileged VFs, so certain network management activities may be conducted within virtual machines rather than only on the hypervisor.

Granting SMP Capability to a Virtual Function

To enable SMP capability for a VF, one must enable the Subnet Management Interface (SMI) for that VF. By default, the SMI interface is disabled for VFs. To enable SMI mads for VFs, there are two new sysfs entries per VF per on the Hypervisor (under `/sys/class/infiniband/mlx4_X/iov/<b.d.f>/ports/<1 or 2>`. These entries are displayed only for VFs (not for the PF), and only for IB ports (not ETH ports).

The first entry, `enable_smi_admin`, is used to enable SMI on a VF. By default, the value of this entry is zero (disabled). When set to "`1`", the SMI will be enabled for the VF on the next rebind or openibd restart on the VM that the VF is bound to. If the VF is currently bound, it must be unbound and then re-bound.

The second sysfs entry, `smi_enabled`, indicates the current enablement state of the SMI. 0 indicates disabled, and 1 indicates enabled. This entry is read-only.

When a VF is initialized (bound), during the initialization sequence, the driver copies the requested `smi_state` (`enable_smi_admin`) for that VF/port to the operational SMI state (`smi_enabled`) for that VF/port, and operate according to the operational state.

Thus, the sequence of operations on the hypevisor is:

Step 1. Enable SMI for any VF/port that you wish.

Step 2. Restart the VM that the VF is bound to (or just run `/etc/init.d/openibd restart` on that VM)

The SMI will be enabled for the VF/port combinations that you set in step 2 above. You will then be able to run network diagnostics from that VF.

Installing MLNX_OFED with Network Diagnostics on a VM

➢ ***To install mlnx_ofed on a VF which will be enabled to run the tools, run the following on the VM:***

```
# mlnx_ofed_install
```

3.4.1.8 MAC Forwarding DataBase (FDB) Management

3.4.1.8.1 FDB Status Reporting

FDB also know as Forwarding Information Base (FIB) or the forwarding table, is most commonly used in network bridging, routing, and similar functions to find the proper interface to which the input interface should forward a packet.

In the SR-IOV environment, the Ethernet driver can share the existing 128 MACs (for each port) among the Virtual interfaces (VF) and Physical interfaces (PF) that share the same table as follow:

- Each VF gets 2 granted MACs (which are taken from the general pool of the 128 MACs)
- Each VF/PF can ask for up to 128 MACs on the policy of first-asks first-served (meaning, except for the 2 granted MACs, the other MACs in the pool are free to be asked)

To check if there are free MACs for its interface (PF or VF), run: `/sys/class/net/<ethX>/fdb_det`.

Example:

```
cat /sys/class/net/eth2/fdb_det
device eth2: max: 112, used: 2, free macs: 110
```

➢ ***To add a new MAC to the interface:***

```
echo +<MAC> > /sys/class/net/eth<X>/fdb
```

Once running the command above, the interface (VF/PF) verifies if a free MAC exists. If there is a free MAC, the VF/PF takes it from the global pool and allocates it. If there is no free MAC, an error is returned notifying the user of lack of MACs in the pool.

➢ ***To delete a MAC from the interface:***

```
echo -<MAC> > /sys/class/net/eth<X>/fdb
```

If `/sys/class/net/eth<X>/fdb` does not exist, use the Bridge tool from the ip-route2 package which includes the tool to manage FDB tables as the kernel supports FDB callbacks:

```
bridge fdb add 00:01:02:03:04:05 permanent self dev p3p1
bridge fdb del 00:01:02:03:04:05 permanent self dev p3p1
bridge fdb show dev p3p1
```

If adding a new MAC from the kernel's NDO function fails due to insufficient MACs in the pool, the following error flow will occur:

- If the interface is a PF, it will automatically enter the promiscuous mode
- If the interface is a VF, it will try to enter the promiscuous mode and since it does not support it, the action will fail and an error will be printed in the kernel's log

3.4.1.9 Virtualized QoS per VF (Rate Limit per VF)

Virtualized QoS per VF, (supported in ConnectX®-3/ConnectX®-3 Pro adapter cards only with firmware v2.33.5100 and above), limits the chosen VFs' throughput rate limitations (Maximum throughput). The granularity of the rate limitation is 1Mbits.

The feature is disabled by default. To enable it, set the “`enable_vfs_qos`” module parameter to “`1`” and add it to the "`options mlx4_core`". When set, and when feature is supported, it will be shown upon PF driver load time (in DEV_CAP in kernel log: *Granular QoS Rate limit per VF support*), when `mlx4_core` module parameter `debug_level` is set to 1. For further information, please refer to Section 1.4.1.2, “mlx4_core Parameters”, on page 25 - debug_level parameter).

When set, and supported by the firmware, running as SR-IOV Master and Ethernet link, the driver also provides information on the number of total available vPort Priority Pair (VPPs) and how many VPPs are allocated per priority. All the available VPPs will be allocated on priority 0.

```
mlx4_core 0000:1b:00.0: Port 1 Available VPPs 63
mlx4_core 0000:1b:00.0: Port 1 UP 0 Allocated 63 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 1 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 2 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 3 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 4 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 5 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 6 Allocated 0 VPPs
mlx4_core 0000:1b:00.0: Port 1 UP 7 Allocated 0 VPPs
```

3.4.1.9.1 Configuring rate limit for VFs

Please note, the rate limit configuration will take effect only when the VF is in VST mode configured with priority 0.

Rate limit can be configured using the iproute2/netlink tool.

```
ip link set dev <PF device> vf <NUM> rate <TXRATE>
```

where

- NUM = 0...<Num of VF>

- <TXRATE> in units of 1Mbit/s

The rate limit for VF can be configured:

- while setting it to the VST mode

  ```
  ip link set dev <PF device> vf <NUM> vlan <vlan_id> [qos <qos>] rate <TXRATE>
  ```

- before the VF enters the VST mode with a supported priority

  In this case, the rate limit value is saved and the rate limit configuration is applied when VF state is changed to VST mode.

To disable rate limit configured for a VF set the VF with rate 0. Once the rate limit is set, you cannot switch to VGT or change VST priority.

To view current rate limit configurations for VFs, use the iproute2 tool.

```
ip link show dev <PF device>
```

Example:

```
89: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether f4:52:14:5e:be:20 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 2, tx rate 1500 (Mbps), spoof checking off, link-state auto
    vf 1 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state auto
    vf 2 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state auto
    vf 3 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-state auto
```

On some OSs, the iptool may not display the configured rate, or any of the VF information, although the both the VST and the rate limit are set through the netlink command. In order to view the rate limit configured, use sysfs provided by the driver. Its location can be found at:

```
/sys/class/net/<eth-x>/<vf-i>/tx_rate
```

3.4.2 Enabling Para Virtualization

➤ ***To enable Para Virtualization:***

Step 1. Create a bridge.

```
vim /etc/sysconfig/network-scripts/ifcfg-bridge0
DEVICE=bridge0
TYPE=Bridge
IPADDR=12.195.15.1
NETMASK=255.255.0.0
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
DELAY=0
```

Step 2. Change the related interface (in the example below bridge0 is created over eth5).

```
DEVICE=eth5
BOOTPROTO=none
STARTMODE=on
HWADDR=00:02:c9:2e:66:52
TYPE=Ethernet
NM_CONTROLLED=no
ONBOOT=yes
BRIDGE=bridge0
```

Step 3. Restart the service network.

Step 4. Attach a virtual NIC to VM.

```
ifconfig -a
…
eth6      Link encap:Ethernet  HWaddr 52:54:00:E7:77:99
          inet addr:13.195.15.5  Bcast:13.195.255.255  Mask:255.255.0.0
          inet6 addr: fe80::5054:ff:fee7:7799/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:481 errors:0 dropped:0 overruns:0 frame:0
          TX packets:450 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:22440 (21.9 KiB)  TX bytes:19232 (18.7 KiB)
          Interrupt:10 Base address:0xa000
…
```

3.4.3 VXLAN Hardware Stateless Offloads

VXLAN technology introduced for solving scalability and security challenges, requires extension of the traditional stateless offloads to avoid performance drop. ConnectX-3 Pro adapter card offers the following stateless offloads for a VXLAN packet, similar to the ones offered to non-encapsulated packets. VXLAN protocol encapsulates its packets using outer UDP header.

Available hardware stateless offloads:

- Checksum generation (Inner IP and Inner TCP/UDP)
- Checksum validation (Inner IP and Inner TCP/UDP). This will allow the use of GRO for inner TCP packets.
- TSO support for inner TCP packets
- RSS distribution according to inner packets attributes
- Receive queue selection - inner frames may be steered to specific QPs

3.4.3.1 Prerequisites

- HCA: ConnectX-3 Pro
- Firmware 2.32.5100 or higher
- RHEL7, Ubuntu 14.04 or upstream kernel 3.12.10 (or higher)
- DMFS enabled
- A0 static mode disabled

3.4.3.2 Enabling VXLAN Hardware Stateless Offloads

To enable the VXLAN offloads support load the `mlx4_core` driver with Device-Managed Flow-steering (DMFS) enabled. DMFS is the default steering mode.

➢ ***To verify it is enabled by the adapter card:***

Step 1. Open the `/etc/modprobe.d/mlnx.conf` file.

Step 2. Set the parameter `debug_level` to `"1"`.

```
options mlx4_core debug_level=1
```

Step 3. Restart the driver.

Step 4. Verify in the dmesg that the tunneling mode is: `vxlan`.

The net-device will advertise the `tx-udp-tnl-segmentation` flag shown when running `"etht-hool -k $DEV | grep udp"` only when VXLAN is configured in the OpenvSwitch (OVS) with the configured UDP port.

For example:

```
$ ethtool -k eth0 | grep udp_tnl
tx-udp_tnl-segmentation: on
```

As of firmware version 2.31.5050, VXLAN tunnel can be set on any desired UDP port. If using previous firmware versions, set the VXLAN tunnel over UDP port 4789.

➤ ***To add the UDP port to /etc/modprobe.d/vxlan.conf:***

```
options vxlan udp_port=<number decided above>
```

3.4.3.3 Important Notes

- VXLAN tunneling adds 50 bytes (14-eth + 20-ip + 8-udp + 8-vxlan) to the VM Ethernet frame. Please verify that either the MTU of the NIC who sends the packets, e.g. the VM virtio-net NIC or the host side veth device or the uplink takes into account the tunneling overhead. Meaning, the MTU of the sending NIC has to be decremented by 50 bytes (e.g 1450 instead of 1500), or the uplink NIC MTU has to be incremented by 50 bytes (e.g 1550 instead of 1500)
- From upstream 3.15-rc1 and onward, it is possible to use arbitrary UDP port for VXLAN. Note that this requires firmware version 2.31.2800 or higher. Additionally, you need to enable this kernel configuration option `CONFIG_MLX4_EN_VXLAN=y`.
- On upstream kernels 3.12/3.13 GRO with VXLAN is not supported

3.5 Resiliency

3.5.1 Reset Flow

Reset Flow is activated by default, once a "fatal device[1]" error is recognized. Both the HCA and the software are reset, the ULPs and user application are notified about it, and a recovery process is performed once the event is raised. The "Reset Flow" is activated by the mlx4_core module parameter `'internal_err_reset'`, and its default value is 1.

3.5.1.1 Kernel ULPs

Once a "fatal device" error is recognized, an `IB_EVENT_DEVICE_FATAL` event is created, ULPs are notified about the incident, and outstanding WQEs are simulated to be returned with `"flush in error"` message to enable each ULP to close its resources and not get stuck via calling its `"remove_one"` callback as part of "Reset Flow".

Once the unload part is terminated, each ULP is called with its `"add_one"` callback, its resources are re-initialized and it is re-activated.

1. A “fatal device” error can be a timeout from a firmware command, an error on a firmware closing command, communication channel not being responsive in a VF. etc.

3.5.1.2 User Space Applications (IB/RoCE)

Once a "fatal device" error is recognized an `IB_EVENT_DEVICE_FATAL` event is created, applications are notified about the incident and relevant recovery actions are taken.

Applications that ignore this event enter a zombie state, where each command sent to the kernel is returned with an error, and no completion on outstanding WQEs is expected.

The expected behavior from the applications is to register to receive such events and recover once the above event is raised. Same behavior is expected in case the NIC is unbounded from the PCI and later is rebounded. Applications running over RDMA CM should behave in the same manner once the `RDMA_CM_EVENT_DEVICE_REMOVAL` event is raised.

The below is an example of using the unbind/bind for NIC defined by "0000:04:00.0"

```
echo 0000:04:00.0 > /sys/bus/pci/drivers/mlx4_core/unbind
echo 0000:04:00.0 > /sys/bus/pci/drivers/mlx4_core/bind
```

3.5.1.3 SR-IOV

If the Physical Function recognizes the error, it notifies all the VFs about it by marking their communication channel with that information, consequently, all the VFs and the PF are reset.

If the VF encounters an error, only that VF is reset, whereas the PF and other VFs continue to work unaffected.

3.5.1.4 Forcing the VF to Reset

If an outside "reset" is forced by using the PCI sysfs entry for a VF, a reset is executed on that VF once it runs any command over its communication channel.

For example, the below command can be used on a hypervisor to reset a VF defined by 0000\:04\:00.1:

```
echo 1 >/sys/bus/pci/devices/0000\:04\:00.1/reset
```

3.5.1.5 Advanced Error Reporting (AER)

AER, a mechanism used by the driver to get notifications upon PCI errors, is supported only in native mode, ULPs are called with `remove_one/add_one` and expect to continue working properly after that flow.User space application will work in same mode as defined in the "Reset Flow" above. Supported in ConnectX-3 and ConnectX-3 Pro only.

3.5.1.6 Extended Error Handling (EEH)

Extended Error Handling (EEH) is a PowerPC mechanism that encapsulates AER, thus exposing AER events to the operating system as EEH events.

The behavior of ULPs and user space applications is identical to the behavior of AER.

3.6 HPC-X™

For further information, please refer to HPC-X™ User Manual (www.mellanox.com --> Products --> HPC-X --> HPC-X Toolkit)

4 InfiniBand Fabric Utilities

This section first describes common configuration, interface, and addressing for all the tools in the package.

4.1 Common Configuration, Interface and Addressing

4.1.1 Topology File (Optional)

An InfiniBand fabric is composed of switches and channel adapter (HCA/TCA) devices. To identify devices in a fabric (or even in one switch system), each device is given a GUID (a MAC equivalent). Since a GUID is a non-user-friendly string of characters, it is better to alias it to a meaningful, user-given name. For this objective, the IB Diagnostic Tools can be provided with a “topology file”, which is an optional configuration file specifying the IB fabric topology in user-given names.

For diagnostic tools to fully support the topology file, the user may need to provide the local system name (if the local hostname is not used in the topology file).

To specify a topology file to a diagnostic tool use one of the following two options:

1. On the command line, specify the file name using the option ‘`-t <topology file name>`’
2. Define the environment variable IBDIAG_TOPO_FILE

To specify the local system name to an diagnostic tool use one of the following two options:

1. On the command line, specify the system name using the option ‘`-s <local system name>`’
2. Define the environment variable IBDIAG_SYS_NAME

4.2 InfiniBand Interface Definition

The diagnostic tools installed on a machine connect to the IB fabric by means of an HCA port through which they send MADs. To specify this port to an IB diagnostic tool use one of the following options:

1. On the command line, specify the port number using the option ‘-p <local port number>’ (see below)
2. Define the environment variable IBDIAG_PORT_NUM

In case more than one HCA device is installed on the local machine, it is necessary to specify the device’s index to the tool as well. For this use on of the following options:

1. On the command line, specify the index of the local device using the following option:
   ‘-i <index of local device>’
2. Define the environment variable IBDIAG_DEV_IDX

4.3 Addressing

This section applies to the ibdiagpath tool only. A tool command may require defining the destination device or port to which it applies.

The following addressing modes can be used to define the IB ports:

- Using a Directed Route to the destination: (Tool option '-d')

  This option defines a directed route of output port numbers from the local port to the destination.
- Using port LIDs: (Tool option '-l'):

  In this mode, the source and destination ports are defined by means of their LIDs. If the fabric is configured to allow multiple LIDs per port, then using any of them is valid for defining a port.
- Using port names defined in the topology file: (Tool option '-n')

  This option refers to the source and destination ports by the names defined in the topology file. (Therefore, this option is relevant only if a topology file is specified to the tool.) In this mode, the tool uses the names to extract the port LIDs from the matched topology, then the tool operates as in the '-l' option.

4.4 Diagnostic Utilities

The diagnostic utilities described in this chapter provide means for debugging the connectivity and status of InfiniBand (IB) devices in a fabric.

Table 5 - Diagnostic Utilities

Utility	Description
dump_fts	Dumps tables for every switch found in an ibnetdiscover scan of the subnet. The dump file format is compatible with loading into OpenSM using the -R file -U /path/to/dump-file syntax. For further information, please refer to the tool's man page.
ibaddr	Can be used to show the LID and GID addresses of the specified port or the local port by default. This utility can be used as simple address resolver. For further information, please refer to the tool's man page.
ibcacheedit	Allows users to edit an ibnetdiscover cache created through the --cache option in ibnetdiscover(8). For further information, please refer to the tool's man page.
ibccconfig	Supports the configuration of congestion control settings on switches and HCAs. For further information, please refer to the tool's man page.
ibccquery	Supports the querying of settings and other information related to congestion control. For further information, please refer to the tool's man page.

Utility	Description
ibcongest	Provides static congestion analysis. It calculates routing for a given topology (topo-mode) or uses extracted lst/fdb files (lst-mode). Additionally, it analyzes congestion for a traffic schedule provided in a "schedule-file" or uses an automatically generated schedule of all-to-all-shift. To display a help message which details the tool's options, please run `"/opt/ibutils2/bin/ibcongest -h"`. For further information, please refer to the tool’s man page.
ibdev2netdev	Enables association between IB devices and ports and the associated net device. Additionally it reports the state of the net device link. For further information, please refer to the tool’s man page.
ibdiagnet **(of ibutils)**	This version of ibdiagnet is included in the ibutils package, and it is not run by default after installing Mellanox OFED. To use this ibdiagnet version and not that of the ibutils package, you need to specify the full path: `/opt/ibutils/bin` **Note:** ibdiagnet is an obsolete package. We recommend using ibdiagnet from ibutils2. For further information, please refer to the tool’s man page.
ibdiagnet **(of ibutils2)**	Scans the fabric using directed route packets and extracts all the available information regarding its connectivity and devices. An ibdiagnet run performs the following stages: • Fabric discovery • Duplicated GUIDs detection • Links in INIT state and unresponsive links detection • Counters fetch • Error counters check • Routing checks • Link width and speed checks • Alias GUIDs check • Subnet Manager check • Partition keys check • Nodes information **Note:** This version of ibdiagnet is included in the ibutils2 package, and it is run by default after installing Mellanox OFED. To use this ibdiagnet version, run: `ibdiagnet`. For further information, please refer to the tool’s man page.

Utility	Description
ibdiagpath	Traces a path between two end-points and provides information regarding the nodes and ports traversed along the path. It utilizes device specific health queries for the different devices along the path. The way ibdiagpath operates depends on the addressing mode used on the command line. If directed route addressing is used (-d flag), the local node is the source node and the route to the destination port is known apriori. On the other hand, if LID-route (or by-name) addressing is employed, then the source and destination ports of a route are specified by their LIDs (or by the names defined in the topology file). In this case, the actual path from the local port to the source port, and from the source port to the destination port, is defined by means of Subnet Management Linear Forwarding Table queries of the switch nodes along that path. Therefore, the path cannot be predicted as it may change. ibdiagpath should not be supplied with contradicting local ports by the -p and -d flags (see synopsis descriptions below). In other words, when ibdiagpath is provided with the options -p and -d together, the first port in the direct route must be equal to the one specified in the “-p” option. Otherwise, an error is reported. Moreover, the tool allows omitting the source node in LID-route addressing, in which case the local port on the machine running the tool is assumed to be the source. **Note:** When ibdiagpath queries for the performance counters along the path between the source and destination ports, it always traverses the LID route, even if a directed route is specified. If along the LID route one or more links are not in the ACTIVE state, ibdiagpath reports an error. ibdiagpath is located at: `/opt/ibutisl/bin`. For further information, please refer to the tool’s man page.
ibdump	Dump InfiniBand traffic that flows to and from Mellanox Technologies ConnectX® family adapters InfiniBand ports. The dump file can be loaded by the Wireshark tool for graphical traffic analysis. The following describes a work flow for local HCA (adapter) sniffing: 1. Run ibdump with the desired options 2. Run the application that you wish its traffic to be analyzed 3. Stop ibdump (CTRL-C) or wait for the data buffer to fill (in --mem-mode) 4. Open Wireshark and load the generated file To download Wireshark for a Linux or Windows environment go to www.wireshark.org. **Note:** Although ibdump is a Linux application, the generated .pcap file may be analyzed on either operating system. In order for ibdump to function with RoCE, Flow Steering must be enabled. To do so: 1. Add the following to `/etc/modprobe.d/mlnx.conf` file: `options mlx4_core log_num_mgm_entry_size=-1` 2. Restart the drivers. **Note:** If one of the HCA's port is configured as InfiniBand, ibdump requires IPoIB DMFS to be enabled. For further information, please refer to Section 3.1.11.1, “Enable/Disable Flow Steering”, on page 80 For further information, please refer to the tool’s man page.

Utility	Description
iblinkinfo	Reports link info for each port in an InfiniBand fabric, node by node. Optionally, iblinkinfo can do partial scans and limit its output to parts of a fabric. For further information, please refer to the tool's man page.
ibnetdiscover	Performs InfiniBand subnet discovery and outputs a human readable topology file. GUIDs, node types, and port numbers are displayed as well as port LIDs and node descriptions. All nodes (and links) are displayed (full topology). This utility can also be used to list the current connected nodes. The output is printed to the standard output unless a topology file is specified. For further information, please refer to the tool's man page.
ibnetsplit	Automatically groups hosts and creates scripts that can be run in order to split the network into sub-networks containing one group of hosts. For further information, please refer to the tool's man page.
ibnodes	Uses the current InfiniBand subnet topology or an already saved topology file and extracts the InfiniBand nodes (CAs and switches). For further information, please refer to the tool's man page.
ibping	Uses vendor mads to validate connectivity between InfiniBand nodes. On exit, (IP) ping like output is show. ibping is run as client/server. The default is to run as client. Note also that a default ping server is implemented within the kernel. For further information, please refer to the tool's man page.
ibportstate	Enables querying the logical (link) and physical port states of an InfiniBand port. It also allows adjusting the link speed that is enabled on any InfiniBand port. If the queried port is a switch port, then ibportstate can be used to: • disable, enable or reset the port • validate the port's link width and speed against the peer port In case of multiple channel adapters (CAs) or multiple ports without a CA/port being specified, a port is chosen by the utility according to the following criteria: • The first ACTIVE port that is found. • If not found, the first port that is UP (physical link state is LinkUp). For further information, please refer to the tool's man page.
ibqueryerrors	The default behavior is to report the port error counters which exceed a threshold for each port in the fabric. The default threshold is zero (0). Error fields can also be suppressed entirely. In addition to reporting errors on every port, ibqueryerrors can report the port transmit and receive data as well as report full link information to the remote port if available. For further information, please refer to the tool's man page.
ibroute	Uses SMPs to display the forwarding tables—unicast (LinearForwardingTable or LFT) or multicast (MulticastForwardingTable or MFT)—for the specified switch LID and the optional lid (mlid) range. The default range is all valid entries in the range 1 to FDBTop. For further information, please refer to the tool's man page.

Utility	Description
ibstat	ibstat is a binary which displays basic information obtained from the local IB driver. Output includes LID, SMLID, port state, link width active, and port physical state. For further information, please refer to the tool’s man page.
ibstatus	Displays basic information obtained from the local InfiniBand driver. Output includes LID, SMLID, port state, port physical state, port width and port rate. For further information, please refer to the tool’s man page.
ibswitches	Traces the InfiniBand subnet topology or uses an already saved topology file to extract the InfiniBand switches. For further information, please refer to the tool’s man page.
ibsysstat	Uses vendor mads to validate connectivity between InfiniBand nodes and obtain other information about the InfiniBand node. ibsysstat is run as client/server. The default is to run as client. For further information, please refer to the tool’s man page.
ibtopodiff	Compares a topology file and a discovered listing ofsubnet.lst/ibdiagnet.lst and reports missmatches. Two different algorithms provided: • Using the -e option is more suitible for MANY mismatches it applies less heuristics and provide details about the match • Providing the -s, -p and -g starts a detailed heuristics that should be used when only small number of changes are expected For further information, please refer to the tool’s man page.
ibtracert	Uses SMPs to trace the path from a source GID/LID to a destination GID/LID. Each hop along the path is displayed until the destination is reached or a hop does not respond. By using the -m option, multicast path tracing can be performed between source and destination nodes. For further information, please refer to the tool’s man page.
ibv_asyncwatch	Display asynchronous events forwarded to userspace for an InfiniBand device. For further information, please refer to the tool’s man page.
ibv_devices	Lists InfiniBand devices available for use from userspace, including node GUIDs. For further information, please refer to the tool’s man page.
ibv_devinfo	Queries InfiniBand devices and prints about them information that is available for use from userspace. For further information, please refer to the tool’s man page.
mstflint	Queries and burns a binary firmware-image file on non-volatile (Flash) memories of Mellanox InfiniBand and Ethernet network adapters. The tool requires root privileges for Flash access. To run mstflint, you must know the device location on the PCI bus. **Note:** If you purchased a standard Mellanox Technologies network adapter card, please download the firmware image from `www.mellanox.com > Support > Firmware Download`. If you purchased a non-standard card from a vendor other than Mellanox Technologies, please contact your vendor. For further information, please refer to the tool’s man page.

Utility	Description
perfquery	Queries InfiniBand ports' performance and error counters. Optionally, it displays aggregated counters for all ports of a node. It can also reset counters after reading them or simply reset them. For further information, please refer to the tool's man page.
saquery	Issues the selected SA query. Node records are queried by default. For further information, please refer to the tool's man page.
sminfo	Issues and dumps the output of an sminfo query in human readable format. The target SM is the one listed in the local port info or the SM specified by the optional SM LID or by the SM direct routed path. **Note:** Using sminfo for any purpose other than a simple query might result in a malfunction of the target SM. For further information, please refer to the tool's man page.
smparquery	Sends SMP query for adaptive routing and private LFT features. For further information, please refer to the tool's man page.
smpdump	A general purpose SMP utility which gets SM attributes from a specified SMA. The result is dumped in hex by default. For further information, please refer to the tool's man page.
smpquery	Provides a basic subset of standard SMP queries to query Subnet management attributes such as node info, node description, switch info, and port info. For further information, please refer to the tool's man page.

4.4.1 Link Level Retransmission (LLR) in FDR Links

With the introduction of FDR 56 Gbps technology, Mellanox enabled a proprietary technology called LLR (Link Level Retransmission) to improve the reliability of FDR links.

This proprietary LLR technology adds additional CRC checking to the data stream and retransmits portions of packets with CRC errors at the local link level. Customers should be aware of the following facts associated with LLR technology:

- Traditional methods of checking the link health can be masked because the LLR technology automatically fixes errors. The traditional IB symbol error counter will show no errors when LLR is active.
- Latency of the fabric can be impacted slightly due to LLR retransmissions. Traditional IB performance utilities can be used to monitor any latency impact.
- Bandwidth of links can be reduced if cable performance degrades and LLR retransmissions become too numerous. Traditional IB bandwidth performance utilities can be used to monitor any bandwidth impact.

Due to these factors, an LLR retransmission rate counter has been added to the ibdiagnet utility that can give end users an indication of the link health.

➢ ***To monitor LLR retransmission rate:***

1. Run ibdiagnet, no special flags required.
2. If the LLR retransmission rate limit is exceeded it will print to the screen.
3. The default limit is set to 500 and requires further investigation if exceeded.
4. The LLR retransmission rate is reflected in the results file /var/tmp/ibdiagnet2/ibdiagnet2.pm.

The default value of 500 retransmissions/sec has been determined by Mellanox based on the extensive simulations and testing. Links exhibiting a lower LLR retransmission rate should not raise special concern.

4.5 Performance Utilities

The performance utilities described in this chapter are intended to be used as a performance micro-benchmark.

Utility	Description
ib_atomic_bw	Calculates the BW of RDMA Atomic transactions between a pair of machines. One acts as a server and the other as a client. The client RDMA sends atomic operation to the server and calculate the BW by sampling the CPU each time it receive a successful completion. The test supports features such as Bidirectional, in which they both RDMA atomic to each other at the same time, change of MTU size, tx size, number of iteration, message size and more. Using the "-a" flag provides results for all message sizes. For further information, please refer to the tool’s man page.
ib_atomic_lat	Calculates the latency of RDMA Atomic transaction of message_size between a pair of machines. One acts as a server and the other as a client. The client sends RDMA atomic operation and sample the CPU clock when it receives a successful completion, in order to calculate latency. For further information, please refer to the tool’s man page.
ib_read_bw	Calculates the BW of RDMA read between a pair of machines. One acts as a server and the other as a client. The client RDMA reads the server memory and calculate the BW by sampling the CPU each time it receive a successful completion. The test supports features such as Bidirectional, in which they both RDMA read from each other memory's at the same time, change of mtu size, tx size, number of iteration, message size and more. Read is available only in RC connection mode (as specified in IB spec). For further information, please refer to the tool’s man page.
ib_read_lat	Calculates the latency of RDMA read operation of message_size between a pair of machines. One acts as a server and the other as a client. They perform a ping pong benchmark on which one side RDMA reads the memory of the other side only after the other side have read his memory. Each of the sides samples the CPU clock each time they read the other side memory , in order to calculate latency. Read is available only in RC connection mode (as specified in IB spec). For further information, please refer to the tool’s man page.
ib_send_bw	Calculates the BW of SEND between a pair of machines. One acts as a server and the other as a client. The server receive packets from the client and they both calculate the throughput of the operation. The test supports features such as Bidirectional, on which they both send and receive at the same time, change of mtu size, tx size, number of iteration, message size and more. Using the "-a" provides results for all message sizes. For further information, please refer to the tool’s man page.

Utility	Description
ib_send_lat	Calculates the latency of sending a packet in message_size between a pair of machines. One acts as a server and the other as a client. They perform a ping pong benchmark on which you send packet only if you receive one. Each of the sides samples the CPU each time they receive a packet in order to calculate the latency. Using the "-a" provides results for all message sizes. For further information, please refer to the tool's man page.
ib_write_bw	Calculates the BW of RDMA write between a pair of machines. One acts as a server and the other as a client. The client RDMA writes to the server memory and calculate the BW by sampling the CPU each time it receive a successful completion. The test supports features such as Bidirectional, in which they both RDMA write to each other at the same time, change of mtu size, tx size, number of iteration, message size and more. Using the "-a" flag provides results for all message sizes. For further information, please refer to the tool's man page.
ib_write_lat	Calculates the latency of RDMA write operation of message_size between a pair of machines. One acts as a server and the other as a client. They perform a ping pong benchmark on which one side RDMA writes to the other side memory only after the other side wrote on his memory. Each of the sides samples the CPU clock each time they write to the other side memory, in order to calculate latency. For further information, please refer to the tool's man page.
raw_ethernet_bw	Calculates the BW of SEND between a pair of machines. One acts as a server and the other as a client. The server receive packets from the client and they both calculate the throughput of the operation. The test supports features such as Bidirectional, on which they both send and receive at the same time, change of mtu size, tx size, number of iteration, message size and more. Using the "-a" provides results for all message sizes. For further information, please refer to the tool's man page.
raw_ethernet_lat	Calculates the latency of sending a packet in message_size between a pair of machines. One acts as a server and the other as a client. They perform a ping pong benchmark on which you send packet only if you receive one. Each of the sides samples the CPU each time they receive a packet in order to calculate the latency. Using the "-a" provides results for all message sizes. For further information, please refer to the tool's man page.

5 Troubleshooting

You may be able to easily resolve the issues described in this section. If a problem persists and you are unable to resolve it yourself please contact your Mellanox representative or Mellanox Support at support@mellanox.com.

5.1 General Related Issues

Issue	Cause	Solution
The system panics when it is booted with a failed adapter installed.	Malfunction hardware component	1. Remove the failed adapter. 2. Reboot the system.
Mellanox adapter is not identified as a PCI device.	PCI slot or adapter PCI connector dysfunctionality	1. Run `lspci`. 2. Reseat the adapter in its PCI slot or insert the adapter to a different PCI slot. If the PCI slot confirmed to be functional, the adapter should be replaced.
Mellanox adapters are not installed in the system.	Misidentification of the Mellanox adapter installed	Run the command below and check Mellanox's MAC to identify the Mellanox adapter installed. `lspci | grep Mellanox' or 'lspci -d 15b3:` Mellanox MACs start with: 00:02:C9:xx:xx:xx, 00:25:8B:xx:xx:xx or F4:52:14:xx:xx:xx"

5.2 Ethernet Related Issues

Issue	Cause	Solution
No link.	Misconfiguration of the switch port or using a cable not supporting link rate.	• Ensure the switch port is not down • Ensure the switch port rate is configured to the same rate as the adapter's port

Degraded performance is measured when having a mixed rate environment (10GbE, 40GbE and 56GbE).	Sending traffic from a node with a higher rate to a node with lower rate.	Enable Flow Control on both switch's ports and nodes:
		• On the server side run:
		`ethtool -A <interface> rx on tx on`
		• On the switch side run the following command on the relevant interface:
		`send on force` and `receive on force`

Issue	Cause	Solution
No link with break-out cable.	Misuse of the break-out cable or misconfiguration of the switch's split ports	• Use supported ports on the switch with proper configuration. For further information, please refer to the MLNX_OS User Manual. • Make sure the QSFP break-out cable side is connected to the SwitchX.

5.3 InfiniBand Related Issues

Issue	Cause	Solution
The following messages is logged after loading the driver: `multicast join failed with status -22`	Trying to join a multicast group that does not exist or exceeding the number of multicast groups supported by the SM.	If this message is logged often, check for the multicast group's join requirements as the node might not meet them. Note: If this message is logged after driver load, it may safely be ignored.
Unable to stop the driver with the following on screen message: `ERROR: Module <module> is in use`	An external application is using the reported module.	Manually unloading the module using the '`modprobe -r`' command.
Logical link fails to come up while port logical state is **Initializing.**	The logical port state is in the Initializing state while pending the SM for the LID assignment.	1. Verify an SM is running in the fabric. Run '`sminfo`' from any host connected to the fabric. 2. If SM is not running, activate the SM on a node or on managed switch.

5.4 InfiniBand/Ethernet Related Issues

Issue	Cause	Solution
Physical link fails to negotiate to maximum supported rate.	The adapter is running an outdated firmware.	Install the latest firmware on the adapter.
Physical link fails to come up while port physical state is **Polling**.	The cable is not connected to the port or the port on the other end of the cable is disabled.	• Ensure that the cable is connected on both ends or use a known working cable • Check the status of the connected port using the `ibportstate` command and enable it if necessary
Physical link fails to come up while port physical state is **Disabled**.	The port was manually disabled.	Restart the driver: `/etc/init.d/openibd restart`

Issue	Cause	Solution
InfiniBand utilities commands fail to find devices on the system. For example, the `'ibv_devinfo'` command fail with the following output: `Failed to get IB devices list: Function not implemented`	The InfiniBand utilities commands are invoked when the driver is not loaded.	Load the driver: `/etc/init.d/openibd start`

5.5 Installation Related Issues

Issue	Cause	Solution
Driver installation fails.	The install script may fail for the following reasons: • Using an unsupported installation option • Failed to uninstall the previous installation due to dependencies being used • The operating system is not supported • The kernel is not supported. You can run mlnx_add_kernel_support.sh in order to to generate a MLNX-_OFED package with drivers for the kernel • Required packages for installing the driver are missing • Missing kernel backport support for non supported kernel	• Use only supported installation options. The full list of installation options case be displayed on screen by using: `mlnxofedinstall --h` • Run `'rpm -e'` to display a list of all RPMs and then manually uninstall them if the preliminary uninstallation failed due to dependencies being used. • Use a supported operating system and kernel • Manually install the missing packages listed on screen by the installation script if the installation failed due to missing pre-requisites.

Issue	Cause	Solution
After driver installation, the openibd service fail to start. This message is logged by the driver: `Unknown symbol`	The driver was installed on top of an existing In-box driver.	1. Uninstall the MLNX_OFED driver. `ofed_uninstall.sh` 2. Reboot the server. 3. Search for any remaining installed driver. `locate mlx4_ib.ko` `locate mlx4_en.ko` `locate mlx4_core` If found, move them to the `/tmp` directory from the current directory 4. Re-install the MLNX_OFED driver. 5. Restart the openibd service.

5.6 Performance Related Issues

Issue	Cause	Solution
The driver works but the transmit and/or receive data rates are not optimal.		These recommendations may assist with gaining immediate improvement: 1. Confirm PCI link negotiated uses its maximum capability 2. Stop the IRQ Balancer service. `/etc/init.d/irq_balancer stop` 3. Start mlnx_affinity service. `mlnx_affinity start` For best performance practices, please refer to the *"Performance Tuning Guide for Mellanox Network Adapters"* (www.mellanox.com > Products > InfiniBand/VPI Drivers > Linux SW/ Drivers).
Out of the box throughput performance in Ubuntu14.04 is not optimal and may achieve results below the line rate in 40GE link speed.	IRQ affinity is not set properly by the `irq_balancer`	For additional performance tuning, please refer to Performance Tuning Guide.
UDP receiver throughput may be lower then expected, when running over mlx4_en Ethernet driver.	This is caused by the adaptive interrupt moderation routine, which sets high values of interrupt coalescing, causing the driver to process large number of packets in the same interrupt, leading UDP to drop packets due to overflow in its buffers.	Disable adaptive interrupt moderation and set lower values for the interrupt coalescing manually. `ethtool -C <eth>X adaptive-rx off rx-usecs 64 rx-frames 24` Values above may need tuning, depending the system, configuration and link speed.

5.7 SR-IOV Related Issues

Issue	Cause	Solution
Failed to enable SR-IOV. The following message is reported in dmesg: `mlx4_core 0000:xx:xx.0: Failed to enable SR-IOV, continuing without SR-IOV (err = -22)`	The number of VFs configured in the driver is higher than configured in the firmware.	1. Check the firmware SR-IOV configuration, run the mlxconfig tool. 2. Set the same number of VFs for the driver.
Failed to enable SR-IOV. The following message is reported in dmesg: `mlx4_core 0000:xx:xx.0: Failed to enable SR-IOV, continuing without SR-IOV (err = -12)`	SR-IOV is disabled in the BIOS.	Check that the SR-IOV is enabled in the BIOS (see Section 3.4.1.2, “Setting Up SR-IOV”, on page 199).
When assigning a VF to a VM the following message is reported on the screen: `"PCI-assgine: error: requires KVM support"`	SR-IOV and virtualization are not enabled in the BIOS.	1. Verify they are both enabled in the BIOS 2. Add to the GRUB configuration file to the following kernel parameter: `"intel_immun=on"` (see Section 3.4.1.2, “Setting Up SR-IOV”, on page 199).

5.8 PXE (FlexBoot) Related Issues

Issue	Cause	Solution
PXE boot timeout.	The `'always-broadcast'` option is disabled.	Enable `'always-broadcast on'`. For the complete procedure, please refer to 'Linux PXE User Manual'
PXE InfiniBand link fails with the following messages although the DHCP request was sent: *Initializing* and *The socket is not connected.*	Either the SM is not running in the fabric or the SM default multicast group was created with non default settings.	1. Activate the SM on a node or on managed switch. 2. Check in the SM `partitions.conf` file that the default partition rate and MTU setting are SDR and 2K, respectively. The PXE is establishing by default an SDR link set with an MTU of 2K. If the default multicast group opened with different rate and/or MTU, the SM will deny the PXE request to join.

Issue	Cause	Solution
Mellanox adapter is not identified as a boot device.	The expansion ROM image is not installed on the adapter. or the server's BIOS is not configured to work on Legacy mode	1. Run a `flint` query to display the expansion ROM information. For example: `"flint -d /dev/mst/mt4099_pci_cr0 q"` and look for the `"Rom info:"` line. For further information on how to burn the ROM, please refer to MFT User Manual. 2. Make sure the BIOS is configured to work in Legacy mode if the adapter's firmware does not include a UEFI image.

5.9 RDMA Related Issues

Issue	Cause	Solution
Infiniband-diags tests, such as `'ib_write_bw'`, fail between systems with different driver releases.	When running a test between 2 systems in the fabric with different Infiniband-diags packages installed.	Run the test using the same perftest RPM on both systems.

5.10 Debugging Related Issues

Issue	Cause	Solution

False positive errors when running applications with valgrind.	Default MLNX_OFED libraries are compiled without valgrind support and several resources are managed by the kernel.	The following MLNX_OFED libraries are now also compiled with valgrind support:
		• libibverbs
		• libmlnx4
		• libmlx5
		• librdmacm
		Libraries' files compiled with valgrind support are installed under `"/usr/lib64/mlnx_ofed/valgrind/"`
		
		• To run an application over these libraries, thus prevent false positive errors,:
		`# env LD_LIBRARY_PATH=/usr/lib64/mlnx_ofed/valgrind/ valgrind [valgrind options] <application cmd>`
		• To suppress most of valgrind's false positive errors, generate the suppression file:
		`#./generate_mlnx_ofed_supp.sh  > mlnx.supp`